STATA TECHNICAL BULLETIN

A publication to promote communication among Stata users

Subscriptions are available from Stata Corporation, email stata@stata.com, telephone 979-696-4600 or 800-STATAPC, fax 979-696-4601. Current subscription prices are posted at www.stata.com/bookstore/stb.html.

Previous Issues are available individually from StataCorp. See www.stata.com/bookstore/stbj.html for details.

Submissions to the STB, including submissions to the supporting files (programs, datasets, and help files), are on a nonexclusive, free-use basis. In particular, the author grants to StataCorp the nonexclusive right to copyright and distribute the material in accordance with the Copyright Statement below. The author also grants to StataCorp the right to freely use the ideas, including communication of the ideas to other parties, even if the material is never published in the STB. Submissions should be addressed to the Editor. Submission guidelines can be obtained from either the editor or StataCorp.

Contents of this issue

stata49	Interrater agreement

William Gould, Stata Corporation, wgould@stata.com

`kap` has been updated in three ways:

1. In the two-identified rater case, when one rater or the other does not use certain ratings, the output now looks better. This is a purely cosmetic improvement; statistics were always correct but the previous output might have led you to think otherwise.

2. In the two-identified rater case, when both raters do not use certain ratings, a new `absolute` option makes producing weighted kappas easier. As things were before, you could not use the predefined `wgt(w)` or `wgt(w2)` options and, if you wanted to specify a weighting scheme for yourself, you had to make a unique matrix according to the pattern of the data that did exist.

3. `kap` can now be used with three or more raters and with a varying number of raters. The alternative `kappa` command could always handle such cases but `kap` and `kappa` assume the data are organized differently. `kap` assumes the variables record individual ratings. `kappa` assumes variables record the frequencies with which each rating occurred. Converting a `kap`-format dataset into the form required by `kappa` was tedious at best; now that is not necessary.

Syntax

> `kap` *varname₁* *varname₂* [*weight*] [if *exp*] [in *range*] [, absolute tab wgt(*wgtid*)]
>
> `kapwgt` *wgtid* 1 \ # 1 [\ # # 1 ...]
>
> `kap` *varname₁* *varname₂* *varname₃* [...] [*weight*] [if *exp*] [in *range*]
>
> `kappa` *varlist* [if *exp*] [in *range*]

`fweights` are allowed; see [U] **18.1.6 weight**.

Description

`kap` (first syntax) calculates the kappa-statistic measure of interrater agreement when there are two unique raters and two or more ratings. `kapwgt` defines weights for use by `kap` in measuring the importance of disagreements.

`kap` (second syntax) and `kappa` calculate the kappa-statistic measure in the case of two or more (nonunique) raters and two outcomes, more than two outcomes when the number of raters is fixed, and more than two outcomes when the number of raters varies. `kap` (second syntax) and `kappa` produce the same results; they merely assume the data are organized differently.

`kap` assumes each observation is a subject. *varname₁* contains the ratings by the first rater, *varname₂* the ratings by the second rater, and so on.

`kappa` also assumes each observation is a subject. The variables, however, record the frequencies with which ratings were assigned. The first variable in *varlist* records the number of times the first rating was assigned, the second variable the number of times the second rating was assigned, and so on.

Options

`absolute` is relevant only if `wgt()` is also specified; see `wgt()` below. Option `absolute` modifies how i, j, and k in the formulas below are defined and how corresponding entries are found in a user-defined weighting matrix. When `absolute` is not specified, i and j refer to the row and column index, not the ratings themselves. Say the ratings are recorded as $\{0, 1, 1.5, 2\}$. There are 4 ratings; $k = 4$ and i and j are still 1, 2, 3, and 4 in the formulas below. Index 3, for instance, corresponds to rating = 1.5. This is convenient but can, with some data, lead to difficulties.

`tab` displays a tabulation of the assessments by the two raters.

`wgt(`*wgtid*`)` specifies that *wgtid* is to be used to weight disagreements. User-defined weights can be created using `kapwgt`; in that case, `wgt()` specifies the name of the user-defined matrix. For instance, you might define

```
. kapwgt mine 1 \ .8 1 \ 0 .8 1 \ 0 0 .8 1
```

and then

```
. kap rata ratb, wgt(mine)
```

In addition, two prerecorded weights are available.

wgt(w) specifies weights $1 - |i - j|/(k - 1)$, where i and j index the rows and columns of the ratings by the two raters and k is the maximum number of possible ratings.

wgt(w2) specifies weights $1 - ((i - j)/(k - 1))^2$.

Let's consider another example: The recorded ratings are $\{1, 2, 3, 4\}$ but rating $= 3$ was never assigned by either rater. Then kap would determine the ratings are from the set $\{1, 2, 4\}$ because those were the only values observed. kap would expect a user-defined weighting matrix to be 3×3 and, were it not, kap would issue an error message. In the formula-based weights, the calculation would be based on $i, j = 1, 2, 3$ corresponding to the three observed ratings $\{1, 2, 4\}$.

Specifying absolute would make it clear that the ratings are 1, 2, 3, and 4; it just so happens that rating $= 3$ was never assigned. Were a user-defined weighting matrix also specified, kap would expect it to be 4×4 or larger (larger because one can think of the ratings being 1, 2, 3, 4, 5, ... and it just so happens that rating 5, 6, ..., were never observed just as rating $= 3$ was not observed.) In the formula-based weights, the calculation would be based on $i, j = 1, 2, 4$.

When absolute is specified, all ratings must be integers and they must be coded from the set $\{1, 2, 3, \ldots\}$. Not all values need be used; integer values that do not occur are simply assumed to be unobserved.

If all conceivable ratings are observed in the data, then whether absolute is specified makes no difference. For instance, if rater A assigns ratings $\{1, 2, 4\}$ and rater B assigns $\{1, 2, 3, 4\}$, then the complete set of assigned ratings is $\{1, 2, 3, 4\}$, the same as absolute would specify. And without absolute, it makes no difference whether the ratings are $\{1, 2, 3, 4\}$, $\{0, 1, 2, 3\}$, $\{1, 7, 9, 100\}$, $\{0, 1, 1.5, 2.0\}$, or coded any other way.

Example 1: Two raters

```
. kap rata ratb
. kap rata ratb, tabulate
. kap rata ratb, wgt(w)
. kap rata ratb, wgt(w2)
```

In the above examples, each observation in the dataset records the outcome for a single patient. A little bit of the data might be

```
. list patid rata ratb in 1/5
          patid        rat1        rat2
   1.      1046           1           1
   2.       421           3           4
   3.      1107           3           2
   4.      1818           3           2
   5.       554           3           3
```

Example 2: Two raters, data from a table

The following data are given to you on paper:

```
             |  Rater B:
             |  1   2   3
-------------+------------
Rater A:  1  |  6   4   3
          2  |  5   3   3
          3  |  0   0  26
```

The corresponding dataset would be

```
. list
          rata     ratb      pop
   1.       1        1        6
   2.       1        2        4
   3.       1        3        3
   4.       2        1        5
   5.       2        2        3
   6.       2        3        3
   7.       3        1        0
   8.       3        2        0
   9.       3        3       26
```

and the command to produce the kappa statistic is

```
. kap rata ratb [freq=pop]
```

```
              Expected
Agreement     Agreement      Kappa         Z          Pr>Z
-----------------------------------------------------------------
  70.00%        42.08%       0.4820       4.74        0.0000
```

Also see help `tabi`; the easy way to enter this data would be

```
. tabi 6 4 3 \ 5 3 3 \ 0 0 26, replace
```

Example 3: Two raters, weighted kappa

Two raters rate patients into four categories. You want to use the weighting matrix:

```
Rater A |  normal  benign  suspect  cancer
--------+---------------------------------
normal  |    1       .8       0         0
benign  |    .8      1        0         0
suspect |    0       0        1        .8
cancer  |    0       0       .8         1
```

You type

```
. kapwgt xm 1 \ .8 1 \ 0 0 1 \ 0 0 .8 1
```

to define the weighting matrix. You can type `kapwgt xm` to verify that you have entered the matrix correctly:

```
. kapwgt xm
 1.0000
 0.8000 1.0000
 0.0000 0.0000 1.0000
 0.0000 0.0000 0.8000 1.0000
```

You then type

```
. kap rata ratb, wgt(xm)
```

to produce the weighted kappa.

Example 4: Two raters, some ratings unobserved

You have data on individual patients. A summary of the observed ratings is

```
. tabulate rata ratb
         | ratb
    rata |         1         2         4 |     Total
---------+---------------------------------+----------
       1 |         6         4         3 |        13
       2 |         5         3         3 |        11
       4 |         1         1        26 |        28
---------+---------------------------------+----------
   Total |        12         8        32 |        52
```

Note that neither rater ever used the rating 3. Whether you type

```
. kap rata ratb
```

or

```
. kap rata ratb, absolute
```

makes no difference, but the `absolute` option does affect the output of

```
. kap rata ratb, wgt(w)
```

and

```
. kap rata ratb, wgt(w) absolute
```

Similarly, were you to type

```
. kap rata ratb, wgt(mywgt)
```

or

```
. kap rata ratb, wgt(mywgt) absolute
```

the weighting matrix would be required to be 3×3 in the first case and 4×4 or larger in the second.

Example 5: More than two raters, more than two ratings, fixed number of raters

You have data on patients. Variable `cat1` records the number of raters assessing category 1, `cat2` the number assessing category 2, and `cat3` the number assessing category 3. Note the very different structure of this data from that in the previous examples. Variables contain not ratings but frequencies of ratings. A small part of the data is

```
. list in 1/5
        patid     cat1     cat2     cat3
  1.     1039        3        1        0
  2.     1045        1        2        1
  3.     1047        2        1        1
  4.     1048        0        1        3
  5.     1049        1        2        1
```

These data record three ratings because there are three variables: `cat1`, `cat2`, and `cat3` and four raters because `cat1 + cat2 + cat3 = 4` in all observations. To obtain the kappa statistic, you type

```
. kappa cat1 cat2 cat3
```

In this case you use `kappa`, not `kap`.

Example 5′: More than two raters, more than two ratings, fixed number of raters

This example is the same as the previous example except that the data are recorded differently. Observations are patients but this time `rat1` records the first rater's rating, `rat2` the second's, and so on. A small piece of this data is

```
. list in 1/5
        patid     rat1     rat2     rat3     rat4
  1.     1039        3        3        3        2
  2.     1045        2        1        2        3
  3.     1047        1        2        3        1
  4.     1048        2        3        3        3
  5.     1049        3        1        2        2
```

These data record four raters because there are four variables: `rat1`, `rat2`, `rat3`, and `rat4` and three ratings because the set of values recorded in `rat1`, `rat2`, `rat3`, and `rat4` is $\{1, 2, 3\}$. To obtain the kappa statistic, you type

```
. kap rat1 rat2 rat3 rat4
```

The results will be the same as in Example 5. Again, the information of which rater is which is not exploited when there are more than two raters.

Example 6: More than two raters, two ratings

You have data recording, for each patient, the number of positive and number of negative ratings; a small part of the data is

```
. list in 1/5
        patid      pos      neg
  1.     1039        3        0
  2.     1045        1        2
  3.     1047        2        1
  4.     1048        2        0
  5.     1049        0        2
```

`pos` records the number of positive ratings, `neg` the number negative. These data record two ratings because there are two variables: `pos` and `neg` and more than two raters because `pos + neg > 2` in some observations. `pos + neg` is the number of raters which are $3 + 0 = 3$, $1 + 2 = 3$, $2 + 1 = 3$, $2 + 0 = 2$, and $0 + 2 = 2$. Thus, the number of raters vary.

To obtain the kappa statistic, you type

```
. kappa pos neg
```

Example 6′: More than two raters, two ratings

This is the same as Example 6 except the data are recorded differently. This time the first five observations contain

```
. list in 1/5
            patid      rat1      rat2      rat3
     1.      1039         2         2         2
     2.      1045         2         1         1
     3.      1047         1         2         2
     4.      1048         2         .         2
     5.      1049         2         2         .
```

`rat1` records the ratings by rater 1, `rat2` the ratings by rater 2, and `rat3` the ratings by rater 3.

The number of raters varies from observation to observation because `rat1`, `rat2`, and `rat3` sometimes each contain missing values. The number of ratings is 2 because the set of values for the recorded `rat1`, `rat2`, and `rat3` is $\{1, 2\}$. To obtain the kappa statistic, you type

```
. kap rat1 rat2 rat3
```

The results will be the same as in Example 6. Again, the information of which rater is which is not exploited when there are more than two raters.

Example 7: More than two raters, more than two ratings, varying number of raters

This is similar to Example 5, the difference being that the number of raters varies:

```
. list in 1/5
            patid      cat1      cat2      cat3
     1.      1039         2         1         0
     2.      1045         1         2         1
     3.      1047         2         2         1
     4.      1048         0         1         3
     5.      1049         3         2         1
```

`cat1` records the number of raters assessing category 1, `cat2` the number assessing category 2, and `cat3` the number assessing category 3.

These data record three ratings because there are three variables: `cat1`, `cat2`, and `cat3` and a varying number of raters because `cat1` + `cat2` + `cat3` is not constant. To obtain the kappa statistic, you type

```
. kappa cat1 cat2 cat3
```

Kappa will be calculated, but there is no statistic for testing kappa > 0 in this case and so none will be reported.

Example 7′: More than two raters, more than two ratings, varying number of raters

This is the same as Example 7 except that the variables record ratings rather than frequencies of ratings. A portion of the data is

```
. list in 1/5
            patid      rat1      rat2      rat3      rat4      rat5      rat6
     1.      1039         1         1         .         2         .         .
     2.      1045         1         2         2         .         1         .
     3.      1047         1         1         2         3         2         .
     4.      1048         3         2         .         3         .         3
     5.      1049         1         2         1         1         3         2
```

`rat1` records the first rater's rating, `rat2` the second's, and so on.

These data record a varying number of raters because the `rat1`, `rat2`, ..., `rat6` variables sometimes contain missing values. These data record three ratings because the set of values recorded in `rat1`, `rat2`, `rat3`, `rat4`, `rat5`, and `rat6` is $\{1, 2, 3\}$. To obtain the kappa statistic, you type

```
. kap rat1 rat2 rat3 rat4 rat5 rat6
```

The results will be the same as in Example 7. Kappa will be calculated, but there is no statistic for testing kappa > 0 in this case and so none is reported.

Example 8: The absolute option

Two raters evaluate the same set of x-rays which they rate as normal, benign, suspect, or cancerous and which are coded 1, 2, 3, and 4 in our data. A piece of the data is

```
. list in 1/5
          patid      rat1      rat2     group
  1.        106         1         1         1
  2.        112         3         2         1
  3.        113         3         2         1
  4.        114         3         4         1
  5.        122         3         4         1
```

We wish to weight disagreement by

```
Rater A |   normal   benign   suspect   cancer
--------+--------------------------------------
normal  |     1        .8        0         0
benign  |    .8         1        0         0
suspect |     0         0        1        .8
cancer  |     0         0       .8         1
```

so first we define our weighting matrix and then use `kap` to calculate the weighted kappa:

```
. kapwgt xm 1 \ .8 1 \ 0 0 1 \ 0 0 .8 1
. kap rat1 rat2, wgt(xm) tab
           | rat2
      rat1 |         1         2         3         4 |     Total
-----------+--------------------------------------------+----------
         1 |         9         4         0         0 |        13
         2 |         4         7         0         0 |        11
         3 |         0         5         0        10 |        15
         4 |         0         3         0         8 |        11
-----------+--------------------------------------------+----------
     Total |        13        19         0        18 |        50
Ratings weighted by:
    1.0000    0.8000    0.0000    0.0000
    0.8000    1.0000    0.0000    0.0000
    0.0000    0.0000    1.0000    0.8000
    0.0000    0.0000    0.8000    1.0000

              Expected
Agreement    Agreement       Kappa           Z         Pr>Z
------------------------------------------------------------
   76.80%       44.16%       0.5845        5.34       0.0000
```

Note that rater 2 never used rating = 3. That, however, does not matter since rater 1 did use that rating. We would obtain the same output were we to specify the `absolute` option.

The x-rays occur in two groups, 1 and 2. We wish to also examine the agreement for group 2:

```
. kap rat1 rat2 if group==2, wgt(xm) tab
           | rat2
      rat1 |         1         2         4 |     Total
-----------+-----------------------------------+----------
         1 |         3         0         0 |         3
         2 |         0         3         0 |         3
         4 |         0         3         8 |        11
-----------+-----------------------------------+----------
     Total |         3         6         8 |        17
kapwgt not 3 x 3
r(198);
```

In this subgroup of the x-rays, rater 1 also never used rating = 3. Thus, the table is 3×3 and our weighting matrix is 4×4. This is a case when specifying the `absolute` option is necessary:

```
. kap rat1 rat2 if group==2, wgt(xm) tab absolute
```

```
                     |  rat2
              rat1 |         1         2         4 |     Total
           -----------+--------------------------------+----------
                 1 |         3         0         0 |         3
                 2 |         0         3         0 |         3
                 4 |         0         3         8 |        11
           -----------+--------------------------------+----------
             Total |         3         6         8 |        17
       Ratings weighted by:
          1.0000   0.8000   0.0000
          0.8000   1.0000   0.0000
          0.0000   0.0000   1.0000
                       Expected
       Agreement    Agreement      Kappa         Z        Pr>Z
       ---------------------------------------------------------------
         82.35%       47.27%      0.6654       3.18      0.0007
```

Note that the table is still presented as being 3×3 but, if you look carefully, you will also note that the appropriate 3×3 submatrix has been extracted to be used as the weighting matrix.

---

stata50	Changes to ttest and sdtest

William M. Sribney, Stata Corporation, FAX 409-696-4601, tech_support@stata.com

The output of the `ttest` and `sdtest` commands has been changed to display standard deviations. The output for a two-sample t test now looks like

```
. ttest mpg, by(foreign)
Two-sample t test with equal variances

------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
Domestic |      52    19.82692     .657777    4.743297    18.50638    21.14747
 Foreign |      22    24.77273     1.40951    6.611187    21.84149    27.70396
---------+--------------------------------------------------------------------
combined |      74     21.2973    .6725511    5.785503     19.9569    22.63769
---------+--------------------------------------------------------------------
    diff |             -4.945804   1.362162                -7.661225   -2.230384
------------------------------------------------------------------------------
Degrees of freedom: 72

             Ho: mean(Domestic) - mean(Foreign) = diff = 0

     Ha: diff < 0                Ha: diff ~= 0                Ha: diff > 0
       t = -3.6308                t = -3.6308                 t = -3.6308
   P < t =   0.0003          P > |t| =   0.0005           P > t =   0.9997
```

The display of the standard deviations allows one to informally assess how close to equality they are. Of course, a formal test can be done using `sdtest`:

```
. sdtest mpg, by(foreign)
Variance ratio test

------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
Domestic |      52    19.82692     .657777    4.743297    18.50638    21.14747
 Foreign |      22    24.77273     1.40951    6.611187    21.84149    27.70396
---------+--------------------------------------------------------------------
combined |      74     21.2973    .6725511    5.785503     19.9569    22.63769
------------------------------------------------------------------------------

                   Ho: sd(Domestic) = sd(Foreign)

          F(51,21) observed  = F_obs            =    0.515
          F(51,21) lower tail = F_L   = F_obs   =    0.515
          F(51,21) upper tail = F_U   = 1/F_obs =    1.943
    Ha: sd(1) < sd(2)         Ha: sd(1) ~= sd(2)         Ha: sd(1) > sd(2)
  P < F_obs = 0.0275     P < F_L + P > F_U = 0.0763    P > F_obs = 0.9725
```

This update to `sdtest` also fixes a bug that appeared in Stata 5.0 when the output was reformatted. The degrees of freedom for the F statistic were interchanged, causing the one-sided p values to be incorrect. Because of the symmetry of the test, the two-sided p values were, however, correct.

These new versions also include the immediate forms of the commands: `ttesti` and `sdtesti`. More results are also saved in the S_# macros, including the p values and standard deviations. These saved results can be viewed with the `disp_s` command; see [U] **20.6 Accessing results from Stata commands** in the Stata User's Guide for more information.

dm50	Defining variables and recording their definitions

John R. Gleason, Syracuse University, loesljrg@ican.net

It is a common experience to find that the exact definition of some useful variable is uncertain; the experience is especially unpleasant if the variable was the result of a long series of attempts and missteps. Stata's characteristics (see [U] **19.8 Characteristics**) provide a way to record a variable's definition so that it is saved as part of the dataset in which the variable resides. Jeroen Weesie's (1997) excellent suite of commands takes just this approach: `genl` issues a `generate` command, attaches a label to the variable, and records the defining operation in a characteristic associated with the variable; `repl` works in a similar way by applying the `replace` command to an existing variable. This insert offers a command `defv` that takes a different approach to supplying such services.

`defv` defines a variable and documents the operation it performs. In a typical usage, if you type

```
. defv x = invnorm(uniform())
```

`defv` will issue the command

```
. generate x = invnorm(uniform())
```

if the variable `x` does not exist, and the command

```
. replace x = invnorm(uniform())
```

if it does. In either case, `defv` records the command issued in a characteristic associated with the variable `x`. At any later point, you can review the definition of variable `x` with

```
. defv x ?
x:
 1. generate x = invnorm(uniform())
 2. replace x = invnorm(uniform())
```

Unlike Weesie's `genl` and `repl` commands, `defv` does not assign a label to the variable. On the other hand, `defv` records every command it applies to a variable, whereas `genl` and `repl` leave only the most recent definition attached to the variable. `defv` also consumes about one-third less memory than the combination of `genl` and `repl`.

In effect, `defv` is an almost complete substitute for Stata's `generate` and `replace` commands. In the case of a new variable, the syntax is

defv [by *varlist* :] [*type*] *newvar* [:*lblname*] = *exp* [if *exp*] [in *range*]

while in the case of an existing variable the syntax is

defv [by *varlist* :] *oldvar* = *exp* [if *exp*] [in *range*] [, <u>nop</u>romote]

Thus, the syntax of `defv` differs (unavoidably) from that of `generate` and `replace` at just one point: a `by` clause follows the command `defv` rather than appearing as a prefix, as with `generate` and `replace`. (See [R] **generate** for details of the remainder of the syntax.) Functionally, `defv` differs (unavoidably) from `generate` and `replace` only because the text that follows `defv` cannot contain the character '"'. Thus,

```
. replace x = 1 if city=="Houston"
```

is acceptable, but you cannot use

```
. defv x = 1 if city=="Houston"
```

This is a limitation imposed by Stata's macros, one suffered by `genl` and `repl` as well.

Finally, consider the names of the characteristics used to store definitions of variables, a matter where there may be different preferences. By default, `defv` uses the same naming system as Stata's `note` command, so that the definitions it stores appear to have been created with `note`. (see [R] **notes**.) An advantage of this style is that variable definitions will be treated in the same way as other notes; for example, they will be recognized by the `describe` command. In addition, the review feature of `defv`, which has syntax

```
defv oldvar ?
```

will display all of the notes associated with *oldvar*, whether created by `note` or `defv`. In particular, '`defv x ?`' produces the same display as '`note x`' but without loading the 3584 bytes of code required by the `note` command.

On the other hand, some users may prefer to keep variable definitions separate from other notes they record about a variable. But it is unwise to provide this ability through an option because it is too easy to neglect to supply the option. For this reason, characteristic naming for definitions is set 'permanently' by a local macro in `defv`. To switch from the default characteristic naming to an alternative, a user need only alter a comment near the top of the `defv.ado` file.

Remark

`defv` is merely a wrapper for `generate` and `replace`, but from a user's perspective it tends to blur the distinction between `generate` and `replace`. This seems somewhat of a shift away from Stata's traditional style of "prove to me that you mean it" toward one of "don't say it unless you mean it." I confess that this shift troubles me, a little, but I find the result to be convenient.

Acknowledgment

Reference

Weesie, J. 1997. dm43: Automatic recording of definitions. *Stata Technical Bulletin* 35: 6–7.

dm51	Defining and recording variable orderings

John R. Gleason, Syracuse University, loesljrg@ican.net

Stata's `order` and `move` commands make it possible to order the variables in a dataset to suit one's purposes. But it may be desirable to enforce a specific ordering of variables in one situation (say, during data entry or editing with the spreadsheet-style data editor), but switch to a different ordering in another situation (perhaps so that a varlist such as `var1-var2` refers to a particular set of variables). Similarly, if several users need to work with the same dataset at various times, an ordering that suits one user's purposes may be inconvenient for another user.

Stata's characteristics (see [U] **19.8 Characteristics**) provide a way to save arbitrary orderings of variables along with a dataset; once saved, an ordering can be re-established with only a few keystrokes. This insert presents a command `vorder` that makes it easy to define and establish favored variable orderings, as well as to review and erase defined orderings. `vorder` has four modes, selected by its first argument; its syntax has four variations.

To begin, one syntax is

vorder <u>save</u> *ordername* [*varlist*]

which saves the list of variables referenced by *varlist* under the name *ordername*. Note that this command merely *records* a list of variable names in a particular order; it does not rearrange the current variable order. *ordername* is any sequence of characters permitted in a Stata name; only the first 6 characters are significant. *varlist* is any list of variables in the current dataset; if *varlist* is absent, its place is taken by the keyword `_all` so that `vorder save` *ordername* records the current ordering of all variables (see [U] **18.1.1 varlist**; [U] **18.4 varlists**).

To illustrate, consider the familiar automobile data:

```
. use auto, replace
(1978 Automobile Data)

. ds
make       price      mpg        rep78      hdroom     trunk      weight     length
turn       displ      gratio     foreign
```

Imagine that you wish to re-arrange the order of these 12 variables, but that you also want to be able to return (quickly) to the ordering shown above by the `ds` command.

```
. vorder save base
```

captures the current ordering and saves it under the name `base`. More precisely, this command saves the expansion of `_all` in a characteristic named `_dta[VObase]`. That is, an order name actually consists of the letters `VO` followed by up to 6 additional letters, digits, or underscores. But `vorder` strips away the letters `VO` if you supply them, so that you can refer to a named

ordering as either VO*x* or just *x*. Hence, the commands `vorder save base` and `vorder save VObase` have exactly the same effect.

You might now reorder the variables using the `order` or the `move` command. For example:

```
. order length weight turn mpg
. ds
length    weight    turn      mpg       make      price     rep78     hdroom
trunk     displ     gratio    foreign
```

moves the four named variables into the first four positions, and pushes the remaining variables downward in the ordering. To re-order the variables according to the order VObase, use

```
. vorder order base
```

Thus, the second mode switches to a named ordering; its syntax is

vorder <u>order</u> *ordername*

where *ordername* is the name of a previously defined ordering, specified with or without the initial letters VO.

More generally, if it is useful to repeatedly move the variables `length`, `weight`, `turn`, and `mpg` to the top of the ordering and then return to the original order, you can define a second ordering. For example,

```
. vorder save 1 length weight turn mpg
```

stores under the name 1 (actually, VO1) the varlist `length weight turn mpg`. You can then toggle back and forth between the two orderings:

```
. vorder order 1
. ds
length    weight    turn      mpg       make      price     rep78     hdroom
trunk     displ     gratio    foreign
. vorder order VObase
. ds
make      price     mpg       rep78     hdroom    trunk     weight    length
turn      displ     gratio    foreign
```

`vorder` can also display a directory of defined variable orderings; the syntax is

vorder <u>list</u> [*ordername*]

where *ordername* is the name of any defined ordering, specified with or without the initial characters VO. If *ordername* is absent, its place is filled with _all, meaning 'all currently defined variable orderings'. So,

```
. vorder list
Order VObase: make price mpg rep78 hdroom trunk weight length turn displ gratio foreign
Order VO1: length weight turn mpg
```

displays the orderings currently defined in our running example.

Finally, `vorder` can erase defined orderings; the syntax is

vorder <u>drop</u> *ordername*

where *ordername* is either the name of a defined ordering, specified with or without the initial characters VO, or the keyword _all, meaning 'all currently defined variable orderings'. In our running example,

```
. vorder drop 1
```

erases the ordering named VO1 but leaves the ordering VObase intact,

```
. vorder list
Order VObase: make price mpg rep78 hdroom trunk weight length turn displ gratio foreign
```

whereas the command `vorder drop _all` removes all orderings that have been defined by `vorder`.

Remark

`vorder` is just a tool for managing varlists to be processed by the `order` command. In the example above,

```
. vorder save 1 length weight turn mpg
```

merely records a varlist with four elements so that it can easily be passed to `order` at some later point. The expression `vorder order 1` in fact issues the command

```
. order length weight turn mpg
```

Note that these varlists are parsed before they are saved. So, `vorder save 2 trunk-turn` saves the variable names in the range `trunk-turn` of the *current* ordering, and that is the varlist that `order` receives from `vorder order 2`; this may differ from the meaning of `trunk-turn` when `vorder order 2` is given.

Acknowledgment

dm52	Executing a command on a subset of the data

Peter Sasieni, Imperial Cancer Research Fund, London, p.sasieni@icrf.icnet.uk

Syntax

$$\texttt{with} \; \big[\textit{varlist}\big] \; \big[\texttt{if} \; \textit{exp}\big] \; \big[\texttt{in} \; \textit{range}\big] \; \big[\texttt{, \underline{no}save}\big]: \quad \textit{stata\_cmd}$$

Description

`with` temporarily drops all but a subset of the data and carries out the *stata_cmd* on the kept data. Any new variables or changes to the subset of the data resulting from the *stata_cmd* will be saved together with the original data. Essentially `with` does 'keep if *exp* in *range*' followed by 'keep *varlist*'.

`with` is useful when one has a very large dataset and the *stata_cmd* creates many temporary variables. It can also be used with "home-made" programs that do not handle `if` and `in` properly.

The option `nosave` does not attempt to save any variables or changes to the data that might be made by *stata_cmd*. The default is to combine the data after execution of *stata_cmd* with the original data using `merge`.

Note that `with` does not work with `by`.

Examples

```
. with y x if group==3: running y x, gen(yfit)
. with meas* in 14001/15000: for meas*: replace @=@+1
```

gr24.1	Easier bar charts: correction

Nicholas J. Cox, University of Durham, UK, FAX (011)-44-91-374 2456, n.j.cox@durham.ac.uk

The command `vbar` (Cox 1997) has been corrected to improve treatment of missing values, so that missing values are now automatically ignored, and to allow sorting the bars in the order defined by a string variable.

References

Cox, N. J. 1997. gr24: Easier bar charts. *Stata Technical Bulletin* 36: 4–8. Reprinted in *Stata Technical Bulletin Reprints*, vol. 6, pp. 44–50.

gr25.1	Spike plots for histograms, rootograms, and time series plots: update

Nicholas J. Cox, University of Durham, UK, FAX (011)-44-91-374 2456, n.j.cox@durham.ac.uk
Anthony R. Brady, Public Health Laboratory Service Statistics Unit, UK, tbrady@phls.co.uk

The `spikeplt` command (Cox and Brady 1997) has been revised so that graphs plot more quickly and store more compactly. In the previous version, each spike was plotted for each observation in each bin, and not just once for each distinct spike. The redundancy led to unnecessarily slow plotting and to painfully large `.gph` files for large datasets. This has now been fixed.

References

Cox, N. J. and A. R. Brady. 1997. gr25: Spike plots for histograms, rootograms, and time series plots. *Stata Technical Bulletin* 36: 8–11. Reprinted in *Stata Technical Bulletin Reprints*, vol. 6, pp. 50–54.

ip20	Checking for sufficient memory to add variables

Peter Sasieni, Imperial Cancer Research Fund, London, p.sasieni@icrf.icnet.uk

`memchk` is a modification of the defunct `memsize`. It is designed to be used by programmers at the beginning of a program that creates many temporary variables. The programmer states how many new variables of various types (e.g., integers, floats, etc.) will be created and `memchk` checks that there is sufficient memory. If there is sufficient, `memchk` is silent. If not, `memchk` issues an error message and return code 900.

`memchk` is most useful when one has a large dataset and a computer intensive program that generates several temporary variables. If the program includes `memchk` early on, then it will exit early rather than producing the message `no room to add more variables` after running for several seconds or even minutes.

Syntax

memchk [int #] [byte #] [long #] [double #] [str# #]

ip21	Storing commands in the keyboard buffer (Windows and Macintosh only)

Jeroen Weesie, Utrecht University, Netherlands, weesie@weesie.fsw.ruu.nl

In my working style for doing statistical analysis, I am used to writing and adding to increasingly elaborate .do files. At the end of such .do files, I frequently want to probe around with variations of the last commands that were issued in the .do file. Stata sensibly does not store all commands from .do files in its keyboard-buffer for review or replay. So how can we avoid having to key-enter possibly long arcane commands issued via a .do file? The command `buffer` comes in handy here. It uses the undocumented Stata command `push` that "stores" a string in the keyboard buffer for replay. Thus,

```
. push reg y x1-x4
```

stores the command `reg y x1-x4` in the keyboard buffer to be restored via the standard "arrow keys". (The `push` command is used heavily in the Stata tutorial system.) The command `keyb` stores a command in the keyboard buffer and executes it as well. Thus, if the command

```
. keyb reg y x1-x4
```

is contained in a .do file, the command `reg y x1-x4` is executed, and, after termination of the .do file, pressing PgUp restores the full `reg` command. This command can be edited with the normal edit keys and re-executed.

ip22	Parsing options with embedded parentheses

Jeroen Weesie, Utrecht University, Netherlands, weesie@weesie.fsw.ruu.nl

This insert describes a utility that may be of interest to those users of Stata who are involved in more advanced Stata programming. This utility provides a work-around for an unfortunate property of Stata's high-level parsing command: It does not carefully match parentheses. Thus, for instance

```
. graph x yhat, title(Predicted values (N=40)) border
```

will result in the error message ") invalid". The reason is that Stata's parser terminates an argument of an option at the first closing parenthesis. In the example above, the argument of title would be

```
Predicted values (N=40
```

the first closing parenthesis after 40 is seen as a token that ends the argument of `title`, leaving one more closing parenthesis for additional processing. This clearly leads to a syntax error, just like in case you issued a command with an extra parenthesis like

```
. graph x yhat, title(Predicted values) ) border
```

The place to solve this problem is of course in the Stata-code of the parser. I hope that in the next Stata release it will indeed have been remedied. [*Editors note: StataCorp has informed me that this is going to be fixed in the next release.*] For the time being I needed a work around, especially for commands in which I want to have options that allow expressions and commands.

Syntax

parsoptp *optname pstring*

The command parsoptp (PARSing OPTions with Parentheses) should be called with some option name *optname* as its first argument, followed by the command line (note that I didn't include double quotes). parsoptp will scan the command line searching for a string *optname(string)*, taking care to match parentheses. We also check that the string occurs in the options part of input, i.e., the part after a comma that does not belong to an expression. To decide whether we are dealing with options or not, however, we have, again, to be careful about parentheses. parsopt is also aware of brackets; in fact it matches on brackets as well, and ensures that they are properly closed and properly nested with parentheses. Also, we have to be careful to remember that a comma is not a switch from non-options to options, but rather a toggle. parsoptp obeys these rules. Parentheses (and brackets) may be nested to an arbitrary level. Also, if *optname* occurs as an argument of another option, it is not counted as a match. parsoptp also has another feature not supported by the standard Stata high-level parser: options that may be specified both with and without arguments. Thus, parsoptp will trigger on *optname* even if it is not followed by an opening parenthesis, provided, again, that it is not embedded in the argument of another option.

parsoptp returns its results via 5 global variables:

S_1 *optname* or nothing

S_2 argument for *optname*, or nothing

S_3 rest of command line, with *optname* and its argument removed (== "S_4, S_5")

S_4 non-options part of input

S_5 rest of options-part of input

I will illustrate the command parsoptp with an example. We want to write a command hlite that produces two-way scatterplots, while highlighting some points using an expression. For instance,

```
. hlite ll df, hi(chiprob(df,ll)<.05)
```

Normally, one would program this roughly like this:

```
program define hlite
        version 5.0
        local varlist "ex min(2) max(2)"
        local options "hi(str) *"
        parse "`*´"
        tempvar hiy
        local y: word 1 of `varlist´
        gen `hiy´ = `y´ if `hi´
        graph `hiy´ `varlist´, `options´
end
```

This rough code would break on the earlier example. Using parsoptp, we would code this as follows:

```
program define hlite
        version 5.0
        * parse off the option hi()
        parsoptp hi `*´        /* note: no double quotes around `*´ */
        if "$S_2" == "" {       /* hi() not found or without argument */
                di in re "option hi() required"
                exit 198
        }
        local hi    "$S_2"      /* argument */
        local rest "$S_3"       /* `*´, with hi() removed */
        * parse the rest
        local varlist "ex min(2) max(2)"
        local options "*"
        parse "`rest´"
        tempvar hiy
        local y: word 1 of `varlist´
        gen `hiy´ = `y´ if `hi´
        graph `hiy´ `varlist´, `options´
end
```

In some of my programs I want to use parsoptp only if users have installed it—Stata currently doesn't provide an elegant mechanism by which a command can specify the non-Stata commands that are invoked. The following code fragment illustrates how this may be accomplished.

```
program define xyopt
        version 5.0
        local cmd "`*´"
        local varlist " specs"
        local if " specs"
        local in " specs"
        local options " options that never need expressions"
        capt parsoptp
        if _rc ~= 199 {
                parsoptp xy `cmd´
                local xy "$S_2"        /* xy(str) matched on parentheses */
                local cmd "$S_3"       /* cmd, with xy(str) removed */
        }
        else {
                di in bl "Install -parsoptp- (STB-40) if you need real expressions in xy()"
                local options "`options´ xy(str)"
        }
        parse "`cmd´"
         rest-of-command
end
```

Examples

Above we noted that parsoptp returns output via the global macros S_1, S_2, and S_3. In the examples below, we give the contents of the macros as if we typed di "$S_1/$S_2/$S_3" after each invocation to parsoptp.

```
. * -exec- does not occur
. parsoptp exec this is nothing, k(2) x(3)
//this is nothing, k(2) x(3)

. * -exec- without argument
. parsoptp exec this isn´t nothing, exec k(2) x(3)
exec//this isn´t nothing,  k(2) x(3)

. * -exec- with argument
. parsoptp exec this is nothing, k(2) exec(this is the option text) x(3)
exec/this is the option text/this is nothing, k(2)  x(3)

. * -exec- with argument with embedded parentheses
. parsoptp exec nothing, exec(option text (k=2) with embedded parentheses) xy(3)
exec/option text (k=2) with embedded parentheses/nothing,  xy(3)

. * -exec- with argument with more embedded parentheses
. parsoptp exec nothing, exec(text (k=(2/3)) with more embedded parenthesis) x(3)
exec/text (k=(2/3)) with more embedded parenthesis/nothing,  x(3)

. * -xy- with argument containing xy()
. parsoptp xy nothing, xy(text xy(k=(2/3)) with more embedded parenthesis) x(3)
xy/text xy(k=(2/3)) with more embedded parenthesis/nothing,  x(3)

. * -xyz- embedded in argument other option
. parsoptp xyz nothing, s(xyz in other option) x(3)
//nothing, s(xyz in other option) x(3)

. * -xyz- as regular option, and embedded in argument of other option
. parsoptp xyz nothing, s(xyz in other option) xyz(option text) x(3)
xyz/option text/nothing, s(xyz in other option)  x(3)

. * -xyz- as regular option, and embedded with argument in argument of other option
. parsoptp xyz nothing, s(even xyz(s-xyz) n other option) xyz(option text) x(3)
xyz/option text/nothing, s(even xyz(s-xyz) n other option)  x(3)

. * unmatched parentheses
. parsoptp xyz nothing, xyz(unmatched parentheses in option text (k=) x(3)
too few ´)´ or ´]´
r(132);

. * parentheses and brackets are not properly nested
. parsoptp xyz nothing if m[1,1(~=1 xyz(opttext] x(3))
too many or mismatching ´)´ or ´]´
r(132);
```

Acknowledgment

I appreciate suggestions by James Hardin (Stata Corporation) in a discussion on options with embedded parentheses.

sbe13.3	Correction to age-specific reference intervals ("normal ranges")

Eileen Wright, Royal Postgraduate Medical School, UK, ewright@rpms.ac.uk
Patrick Royston, Imperial College School of Medicine, UK, proyston@rpms.ac.uk

The code of `xriml.ado` distributed on the STB-36 diskette contained no carriage return after the final line of the file. This causes Stata to issue the `unexpected end of file` error message.

[*Editors note: Our apologies to Professors Wright and Royston for this oversight on our part.*]

sbe18	Sample size calculations for clinical trials with repeated measures data

Paul Seed, United Medical & Dental Schools, Guy's & St. Thomas's Hospitals, UK

Introduction

Stata's `sampsi` command calculates sample sizes and power for trials comparing single measurements of an outcome between two treatment groups. Various options allow for different levels of alpha and beta, for continuous or binary outcomes, for one-sided or two-sided tests, for unequal sized groups, and for comparing one group with an assumed estimate.

The syntax of `sampsi` is

`sampsi` $\#_1$ $\#_2$ $\left[\right.$, `alpha(#)` `power(#)` `n1(#)` `n2(#)` `ratio(#)` `sd1(#)` `sd2(#)` `onesample` `onesided`$\left.\right]$

$\#_1$ and $\#_2$ set the means or proportions, where used, `sd1` and `sd2` set the standard deviations, `ratio` is the ratio of subjects per group.

A typical use might be

```
. sampsi 132 127,   sd1(15.)
Estimated sample size for two-sample comparison of means
Test Ho: m1 = m2, where m1 is the mean in population 1
                  and m2 is the mean in population 2
Assumptions:
         alpha =   0.0500  (two-sided)
         power =   0.9000
            m1 =      132
            m2 =      127
           sd1 =       15
           sd2 =       15
         n2/n1 =     1.00
Estimated required sample sizes:
            n1 =      190
            n2 =      190
```

This shows that to detect a difference of 5 units ($132 - 127$), with a standard deviation of 15, 190 subjects are needed in each group. The various settings: power, significance level, assumed mean and sd, are all given.

Power calculations are always based on beliefs about what the study might find. Even where there is good data available from past studies, they can only be approximate. As a study can actually have only one sample size, power calculations usually focus on a single endpoint, and a single hoped-for difference between the groups.

In medical statistics, the main application of `sampsi` is for planning randomized controlled trials (RCTs) comparing a standard treatment with an experimental therapy. For simple studies, where only one measurement of the outcome is planned, `sampsi` is very useful.

However, many study designs allow for repeated measurements, typically once or more at baseline (immediately before randomization); and at regular intervals during follow-up (after the start of the study). Depending on the analysis method used, and the correlations between measurements at different time points, there can be a great increase in efficiency from such designs over a simple study with one measurement.

Extensions

Frison & Pocock (1992) discuss three such methods for use in RCTs to compare two treatments using a continuous outcome measured at different times on each patient. Each uses the average of baseline measurements $\bar{x}_0$ and follow-up measurements $\bar{x}_1$:

POST outcome is $\bar{x}_1$ where the analysis is by simple t test.

CHANGE outcome is $\bar{x}_1 - \bar{x}_0$ where the analysis is by simple t test.

ANCOVA outcome is $\bar{x}_1 - \beta\bar{x}_0$ where the β is estimated by analysis of covariance, correcting for the average at baseline.

They give formulas for the decrease in variance of the estimate of treatment effect, depending on the number of measurements p at baseline, and r during follow-up; and on the correlations between measurements at different times. Power calculations are based on estimates of a single variance at all time points, σ^2, and three correlations—between baseline measurements $\bar{\rho}_{PRE}$, between follow-up measures $\bar{\rho}_{POST}$, and between baseline and follow-up $\bar{\rho}_{MIX}$. Each is taken as the average of all correlations in the appropriate submatrix.

Often the three correlations are assumed equal. In data from a number of trials, Frison & Pocock found $\bar{\rho}_{PRE}$ and $\bar{\rho}_{POST}$ typically had values around 0.7, while $\bar{\rho}_{MIX}$ was nearer 0.5. This is consistent with the common finding that measurements closer in time are more strongly related.

The improvements in variance over a study with one measurement are:

$$\text{POST} \qquad \frac{1 + (r-1)\bar{\rho}_{POST}}{r}$$

$$\text{CHANGE} \quad \frac{1 + (r-1)\bar{\rho}_{POST}}{r} + \frac{1 + (p-1)\bar{\rho}_{PRE}}{p} - 2\bar{\rho}_{MIX}$$

$$\text{ANCOVA} \quad \frac{1 + (r-1)\bar{\rho}_{POST}}{r} - \frac{\bar{\rho}^2_{MIX}\, p}{1 + (p-1)\bar{\rho}_{PRE}}$$

ANCOVA will always be the most efficient of the three approaches. β is set so that $\beta\bar{x}_0$ accounts for the largest possible variation of $\bar{x}_1$.

For a study with one measurement each at baseline and follow-up, CHANGE will be more efficient than POST provided $\bar{\rho}_{MIX}$ is more than 0.5.

POST ignores all baseline measurements, which tends to make it unpopular. CHANGE is the method most commonly used. It has obvious advantages over POST, and is easier to understand than ANCOVA. With more than one baseline measurement, there is little to chose between CHANGE and ANCOVA.

Figure 1 shows the numbers of patients required for different numbers of follow-up measurements in a study where the possible treatment effect is 40% of the standard deviation, and all correlations are taken as 0.7. The five strategies are POST ($p = 0$), CHANGE and ANCOVA with $p = 1$, and CHANGE and ANCOVA with $p = 3$.
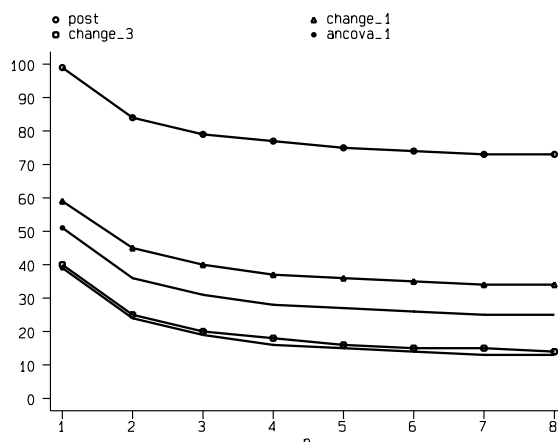

Figure 1: Power calculations for a repeated measures design

sampsi2

I have implemented this in a new command: `sampsi2`

This works as `sampsi`, but with additional arguments to set the method of analysis, the numbers of repeated measurements, and the correlations. These are given in Table 1.

Table 1: additional options for sampsi2

Option	meaning	default value
pre(#)	No. of baseline measurements (pre-randomization)	0
post(#)	No. of follow-up measurements (post-randomization)	1
method(post|change|ancova|all)	method	all
r1(#)	correlation at follow-up	none (r1 must be given)
r0(#)	correlation at baseline	r1
r01(#)	correlation between baseline and follow-up	r1

If any of these options are specified, both `r1` and `sd1` are needed. The output from `sampsi2` includes the settings (including those set by default), the sample sizes and power, the relative efficiency of the design, and the adjustment to the standard deviation. These last two are the inverse and the square root of the values calculated from the variance formulas above.

Sample use

```
. sampsi2 132 127,  pre(1) post(2) sd1(15.) r1(0.7)

Power calculations for repeated measures
(Frison & Pocock 1992)

2 follow-up measurements, correlation  0.700
1 baseline measurement
Correlation between baseline & follow-up  0.700
Raw Standard deviation: 15.000

Method: POST
Relative efficiency:  1.176 Adjustment:  0.922 Adjusted SD: 13.829
Sample sizes: n1 = 161, n2 = 161
significance =  0.050, power =  0.900

Method: CHANGE
Relative efficiency:  2.222 Adjustment:  0.671 Adjusted SD: 10.062
Sample sizes: n1 = 86, n2 = 86
significance =  0.050, power =  0.900

Method: ANCOVA
Relative efficiency:  2.778 Adjustment:  0.600 Adjusted SD:  9.000
Sample sizes: n1 = 69, n2 = 69
significance =  0.050, power =  0.900
```

Stored results

As with `sampsi` S_1 and S_2 contain the sample sizes for the two groups, S_3 the power of the study. In addition, S_4 contains the adjustment to the standard deviations. Results are stored for the last method used.

```
. disp_s
S_1:       69
S_2:       69
S_3:       .9
S_4:       .6
```

Reference

Frison L. and S. Pocock. 1992. Repeated measures in clinical trials: analysis using mean summary statistics and its implications for design. *Statistics in Medicine* 11: 1685–1704.

sg73	Table making programs

John H. Tyler, Harvard Graduate School of Education, tylerjo@hugse1.harvard.edu

The set of (three) "table-maker" programs described in this article produce flexible, user-defined tables of estimation output. Each table can display, in columnar form, the results of up to six (6) different models. When tables produced by these programs are saved in a log file, they can either serve as stand-alone products (e.g., for use in research meetings or for distribution for discussion and comments) or they can be reformatted very easily to generate publishable-quality tables (e.g., for inclusion in

papers). However, while these programs at least partially address a long-standing concern of Stata users regarding the ability to generate estimation output tables, perhaps their greatest value lies in their use as an analytic tool. The programs described in this article allow the data analyst to quickly and easily compare estimates associated with key variables across different models.

The programs are flexible along many dimensions, but a particular strength is that while the default is to display the estimates from all independent variables in the table, the user can easily specify that only a subset of estimates be displayed in the table. Thus, in models where there are some key variables of analytic interest and many control variables, the user can display the results associated with the key analytic variables, suppress the estimates associated with the control variables, and provide wording in the table to indicate that the control variables were included in the just-estimated model. Further options allow the user to choose standard errors or t statistics for display, the number of decimals to be displayed, and whether or not titles for the table and text descriptions of the models in the table are to be displayed.

Two commands, `modl` and `modltbl`, are required to produce an output table following the fitting of one or more models. A third command, `testres`, includes the results of tests of linear constraints that have been conducted following estimation. While detailed syntax and descriptions for each of the commands will follow the examples below, the reader should note that the basic syntax requires following each estimation command with the `modl` command, and then invoking the table with the `modltbl` command. For example,

```
. regress y x1 x2 ...
. modl 1
. regress y x1 x2 ...
. modl 2
. modltbl se 1 2
```

Example 1

A simple example of a table comparing two models is shown below. The example uses the dataset `sg73.dta` included with this insert. It contains information on 37 variables for 4575 individuals, with the dependent variable of interest being `lnearn`, the log of earnings, and the independent variables consisting of various education, work experience, family background, and region of the country. In the baseline model, log wages are regressed on highest grade completed and mother's education. The second model includes dummy variables for race/ethnicity. In this example, t statistics are to be displayed in the table. (The command `quietly` is used in the example, but is not required with the table-maker commands. The table, which would be displayed interactively, begins at the Stata time-date stamp.)

```
. quietly regress lnearn grd10-grd12 momed
. modl a
. quietly regress lnearn grd10-grd12 momed hisp black other
. modl b
. modltbl ts a b

10:47:05 on 20 Oct 1997

(t-statistics in parentheses)
------------------------------
Model :       a           b
# obs :      4575        4575
Depvar:    lnearn      lnearn
------------------------------
intcpt       9.358       9.419
          (129.32)    (129.34)

grd10        0.037       0.036
            (0.55)      (0.55)

grd11        0.134       0.149
            (2.06)      (2.31)

grd12        0.521       0.523
            (9.27)      (9.39)

momed        0.029       0.028
            (7.21)      (6.90)

hisp                    -0.107
                        (-5.00)

black                   -0.208
                        (-7.36)

other                   -0.061
                        (-1.55)

------------------------------
R-sq         0.086       0.099
==============================
```

In the example above, there is little advantage, from an analytic standpoint, to using the table-maker commands. The results from both models will likely fit on one screen, and so, comparisons between the models can be easily made from the standard Stata output. Often, however, the analyst will want to compare several models of increasing complexity or models that have many control variables in addition to the variables of analytic interest. With standard Stata output there are no convenient ways to quickly compare results across models in such cases. The value of the table-maker commands in the face of these situations is evident in the next example.

Example 2

In this example, the interest is in the relationship between log earnings and some key education variables — e10, e11, e12, ged, yrpse, and testsc. The first model in the sequence is the baseline model and contains only the variables of analytic interest. In this case, the default of not specifying any variables in the modl statement will result in display of the estimates associated with all of the independent variables. Hence, the command line

```
. modl 1
```

is entered.

The second model uses global macros to add 27 family background control variables to the baseline model for the estimation. However, in the output table the user wants only the results from the key analytic variables to be displayed. These key variables are the first six independent variables in the regress command. The user also wants it noted in the table that family background controls are used in model number 2, and so the descriptor Fambg is chosen to denote this. The desired output for the second model is accomplished by entering the command line

```
. modl 2 1-6 Fambg
```

The third model adds two work experience controls with a global macro and to indicate this the command line for the third model reads

```
. modl 3 1-6 Fambg Wrkexp
```

The researcher is also interested in displaying the results of two different hypothesis tests. This is done with the testres command.

Models 4, 5, and 6 are simply refits of models 1–3 using robust standard errors.

The modltbl line specifies that t statistics (instead of standard errors) are to be displayed, results are to be displayed to four (4) decimals, models 1 through 6 are to be displayed, and a title is to be added to the table. These results are obtained with the command line

```
. modltbl ts (4) 1 2 3 4 5 6,Table 1. Log earnings regressions with robust SEs in models 4-6
```

Here are the Stata estimation and test commands, the table-maker commands, and the resulting output table:

```
. quietly regress lnearn grd10-grd12 gotged yrcoll testsc
. modl 1
. quietly regress lnearn grd10-grd12 gotged yrcoll testsc $race $region
> momed $dadocc $famin $famstr sibs
. modl 2 1-6 Fambg
. quietly test grd10+gotged=grd12
. testres 2 1, grd10+gotged=grd12
. quietly regress lnearn grd10-grd12 gotged yrcoll testsc $race $region
> momed $dadocc $famin $famstr sibs $work92
. modl 3 1-6 Fambg WrkExp
. quietly test grd10+gotged=grd12
. testres 3 1, grd10+gotged=grd12
. quietly test grd11+gotged=grd12
. testres 3 2, grd11+gotged=grd12
. quietly regress lnearn grd10-grd12 gotged yrcoll testsc,robust
. modl 4
. quietly regress lnearn grd10-grd12 gotged yrcoll testsc $race $region
> momed $dadocc $famin $famstr sibs,robust
. modl 5 1-6 Fambg
. quietly test grd10+gotged=grd12
. testres 5 1, grd10+gotged=grd12
```

```
. quietly regress lnearn grd10-grd12 gotged yrcoll testsc $race $region
> momed $dadocc $famin $famstr sibs $work92,robust
. modl 6 1-6 Fambg WrkExp
. quietly test grd10+gotged=grd12
. testres 6 1, grd10+gotged=grd12
. quietly test grd11+gotged=grd12
. testres 6 2, grd11+gotged=grd12
. modltbl ts (4) 1 2 3 4 5 6, Table 1. Log earnings regressions
> using robust SE's in models 4-6.
10:47:14 on 20 Oct 1997
Table 1. Log earnings regressions using robust SE's in models 4-6.

(t-statistics in parentheses)
---------------------------------------------------------------------------
Model :       1          2          3          4          5          6
# obs :      3928       3928       3928       3928       3928       3928
Depvar:     lnearn     lnearn     lnearn     lnearn     lnearn     lnearn
---------------------------------------------------------------------------
intcpt      9.6210     9.7271     8.4319     9.6210     9.7271     8.4319
          (146.85)    (99.61)    (53.98)   (139.05)    (94.76)    (43.54)

grd10       0.0071    -0.0099    -0.0084     0.0071    -0.0099    -0.0084
           (0.10)     (-0.13)    (-0.12)     (0.09)     (-0.13)    (-0.11)

grd11       0.0538     0.0583     0.0633     0.0538     0.0583     0.0633
           (0.75)      (0.82)     (0.92)     (0.71)      (0.76)     (0.85)

grd12       0.3396     0.3134     0.2536     0.3396     0.3134     0.2536
           (5.07)      (4.68)     (3.90)     (4.80)      (4.34)     (3.65)

gotged      0.1151     0.0964     0.0768     0.1151     0.0964     0.0768
           (2.35)      (1.98)     (1.62)     (2.04)      (1.71)     (1.45)

yrcoll      0.0479     0.0462     0.0621     0.0479     0.0462     0.0621
          (13.27)     (12.42)    (16.42)    (12.49)     (11.60)    (14.67)

testsc      0.0097     0.0076     0.0070     0.0097     0.0076     0.0070
           (9.43)      (7.12)     (6.70)     (9.44)      (7.03)     (6.56)

Fambg                  Yes        Yes                   Yes        Yes
                       ---        ---                   ---        ---

WrkExp                            Yes                              Yes
                                  ---                              ---

---------------------------------------------------------------------------
R-sq        0.175      0.197      0.245      0.175      0.197      0.245
---------------------------------------------------------------------------
Ho_1:,
Pr>F                   0.000      0.000                 0.000      0.000
-------
Ho_2:,
Pr>F                              0.009                            0.025
===========================================================================
Ho_1, F: grd10+gotged=grd12
Ho_2, F: grd11+gotged=grd12
===========================================================================
```

Note that most commercial spreadsheet programs can easily change the text-based table above into a tab-delimited table, which can be exported into virtually all word processing programs for publishable-quality formatting. The rest of this article presents the syntax, descriptions and remarks, and an explanation of options for each of the three table-maker commands modl, modltbl, and testres.

Syntax

modl *model_label* [nocon] [*varlist*] [, *specification*]

where *model_label* is the number or alphanumeric character (of length 1) which labels the model just estimated.

modltbl { ts | se } [(*#decimals* noR2)] *model_label1* [*model_label2 . . . model_label6*] [, *title*]

where ts specifies that t statistics are to be displayed and *model_label* corresponds to some earlier *model_label* specified in a modl command.

testres *model_label* *test_number* [, *text describing the null hypothesis*]

where *model_label* is the label of the model specified in the immediately preceding modl command and *test_number* is a user specified identifier for the just issued test or testparm command.

Description

modl follows any estimation command, and is used in conjunction with modltbl to display the estimated coefficients and t statistics (or standard errors) for selected variables from the just estimated model. modl saves coefficient estimates, standard errors, and t statistics for either all of the independent variables (the default) or for selected variables specified by the arguments following *model_label*. These estimates are saved as global macros which are then available for use by modltbl.

modltbl will compare the estimates on selected variables for up to 6 models. You may, however, have the results from any number of models you have specified with modl stored and available for use by modltbl in any combination. modltbl generates a table of the coefficient estimates and t statistics (or standard errors) for each of the models requested in the command line. Also displayed are a title for the table (optional), the specification of each model (optional), the number of observations used in the estimate, the dependent variable, the R-squared, and the p values on any tests of linear constraints (if a testres command followed any test or testparm command).

testres is used in conjunction with the modl, test (or testparm), and modltbl commands to display in output tables the results (p values) of tests of linear restrictions from estimated models.

Remarks and restrictions for modl

The modl command is the second of three required steps in producing a table which will display the estimates of selected variables from selected models. The steps are (1) estimate a model, (2) immediately follow the estimation command with a modl command, and (3) at any later time in that Stata session invoke display of the desired estimates with a modltbl command. There are certain restrictions which must be kept in mind when using modl. These restrictions are

1. Any *model_label* used in modl must be an alphanumeric character of length 1. That is *model_label* must be of the form 1, 2, 3,..., a, b, c,..., or A, B, C,..., and not 10 or 11 or 1A.

2. modl must immediately follow the estimation command for the model you specify.

3. The variable names used in modl must not be longer than 6 characters. If a variable with a 7 or 8 character name is included as an independent variable in the model the user can rename that "too-long" variable in the modl statement using the newname=oldname option. See the example below under the varlist option.

4. One of the values of modl and modltbl is the ability to limit output in the table to the estimates on user-selected coefficients, while suppressing the output associated with the remaining "control" variables. Indication in the table of the presence of sets of controls can be accomplished through the *Capital-control varlist* option. The restriction is that any *Capital-control* indicator must begin with a capital letter and be no longer than 6 characters. See the example below under the *varlist* option.

5. Note that estimates saved by modl are stored as global macros so that they may be used at any time during a Stata session by modltbl. However, this may cause confusion if you label a model as 1 at one point in the session and then sometime later you estimate another model and also label it as model 1. Note that you can purge all of the saved macros created by various modl commands with the command macro drop _all. Beware, however, that this command will also drop any other global macros you may have created for your own use.

Options for modl

nocon indicates that either (1) the model was fit without a constant or (2) the model may have been fit with a constant, but the user does not wish for the estimates associated with the constant to be displayed in the modltbl table.

varlist can be:

1. a blank space. This is the default setting, and in this case the estimates associated with all of the independent variables in the model are included in the modltbl;

2. _all. This is a second way to capture the estimates of all of the independent variables for display in a table.

3. identification of estimates to be included by number. For example, in a model (say model #1) with many independent variables, the estimates from the 1st–6th, the 8th, and the 10th–12th independent variables could be included in the table by issuing the command:

        . modl 1 1-6 8 10-12

4. identification of estimates to be included by name. For example,

        . modl 1 age gender black

would present the estimates associated with the variables age, gender, and black for model #1 in the table. Note that identification by number and by name can be combined as in:

```
. modl 1 1-6 black
```

5. replacement of a current variable name with a new variable name via the `newname=oldname` option. For example, if a current variable name is more than 6 characters (e.g., `hispanic`), the user can rename this variable in the `modl` statement. This option can be used in combination with any of the other *varlist* options as in:

```
. modl 1 _all hisp=hispanic
. modl 1 1-3 hisp=hispanic
. modl 1 age gender black hisp=hispanic
```

Note that use of this option does not change the actual name of the variable, but simply uses *newvarname* in place of *oldvarname* in the table.

6. *Capital-control* variable sets. In this case the user does not want to display the estimates associated with all of the independent variables included in the just estimated model. The user would, however, like to indicate that a set or sets of controls were included in the model. To do this, append a *Capital-control* indicator at the end of the *varlist*. The restrictions are that the indicator must begin with a capital letter and that as with other variable names, the indicator must be no longer than 6 characters. For example, to indicate that a (potentially long) list of family background and work experience variables were included in the fitting of model #1 issue:

```
. modl 1 1-6 Fambg WrkExp
```

specification allows for the display of a text description of the specification of the model. The text will appear at the top of the output table.

Remarks and restrictions for modltbl

model_label1 model_label2 ... etc., index models identified by various `modl` commands which must have preceded the `modltbl` command. Note that up to six models may be specified for any single `modltbl` command, and that the models need not be specified sequentially. The user may order the models to be displayed in any order in the `modltbl` table.

The following is a list of requirements concerning the use of `modltbl`:

1. `modltbl` works in conjunction with `modl` statements, and the models specified for output in `modltbl` must correspond to model labels from previously executed `modl` statements, each of which must immediately follow a Stata estimation command.

2. If you specify a title, it must be 80 or fewer characters, counting spaces.

Options for modltbl

`ts` or `se` specify whether t statistics or standard errors are to be displayed below each coefficient estimate in the table. Either `ts` or `se` is required as the first argument in `modltbl`.

#decimals is either 2, 3, or 4 and indicates the number of decimal places to display with the coefficient estimates and standard errors. The default is 3, and t statistics are always displayed to two decimal places regardless of the option chosen here.

`noR2` suppresses display of the R-squared statistic in the output. This might be desired, for example, in two-stage least squares regressions.

model_label2 model_label3 ... *model_label6* are the labels attached to each of the additional models the user wishes to display. These model labels are established with the `modl` command.

title is a text description of the table which can be up to 80 characters in length, including spaces.

sg74	Symmetry and marginal homogeneity test / Transmission-Disequilibrium Test (TDT)

<div align="right">Mario Cleves, Stata Corporation, mcleves@stata.com</div>

Syntax

`symmetry` has syntax

    `symmetry` *varcases varcontrols* [*weight*] [`if` *exp*] [`in` *range*] [, <u>notable</u> <u>con</u>trib <u>ex</u>act mh]

    `symmi` $\#_{11}$ $\#_{12}$ [...] \ $\#_{21}$ $\#_{22}$ [...] [\...] [`if` *exp*] [`in` *range*] [, <u>notable</u> <u>con</u>trib <u>ex</u>act mh]

`fweights` are allowed, see [U] **18.1.6 weight**.

Description

`symmetry` performs asymptotic symmetry and marginal homogeneity tests and an exact symmetry test on $K \times K$ tables where there is a 1-to-1 matching of cases and controls (non-independence). This test is used to analyze matched-pair case-control data with multiple discrete levels of the outcome variable. In genetics, the test is known as the Transmission/Disequilibrium test (TDT) and is used to test the association between transmitted and non-transmitted parental marker alleles to an affected child (Spieldman and Ewens 1993). In the case of 2×2 tables the asymptotic test statistics reduce to the McNemar test statistic and the exact symmetry test produces an exact McNemar test.

`symmetry` expects the data to be in the wide format, that is, each observation contains the two matched case and control values in variables *varcases* and *varcontrols*. Variables can be numeric or string.

`symmi` performs the symmetry and marginal homogeneity tests using the values specified on the command line; rows are separated by '\'. The same options as for the `symmetry` command are available for the immediate form. See [U] **25 Immediate commands** for a general description of immediate commands.

Options

`notable` suppresses the output of the contingency table.

`contrib` reports the contribution of each off-diagonal cell-pair to the overall symmetry chi-squared.

`exact` performs an exact test of table symmetry. This option is recommended for sparse tables. CAUTION: the exact test requires substantial amounts of time and computer memory for large tables.

`mh` performs two marginal homogeneity tests that do not require the inversion of the variance–covariance matrix. See *Asymptotic tests* below for a description of these tests.

Asymptotic tests

Consider a square table with K exposure categories, that is, K rows and K columns. Let n_{ij} be the count corresponding to row i and column j of the table, $N_{ij} = n_{ij} + n_{ji}$, for $i, j = 1, 2, \ldots, K$ and $n_{i.}$ and $n_{.j}$ the marginal totals for row i and column j respectively. Asymptotic tests for symmetry and marginal homogeneity for this $K \times K$ table are calculated as follows.

The null hypothesis of complete symmetry $p_{ij} = p_{ji}$ is tested by calculating the test statistic (Bowker 1948):

$$T_{cs} = \sum_{i<j} \frac{(n_{ij} - n_{ji})^2}{n_{ij} + n_{ji}}$$

which is asymptotically distributed as χ^2 with $K(K-1)/2 - R$ degrees of freedom, where R is the number of off-diagonal cells with $N_{ij} = 0$.

The null hypothesis of marginal homogeneity, $H_0 : p_{i.} = p_{.i}$, is tested by calculating the Stuart–Maxwell test statistic (Stuart 1995, Maxwell 1970):

$$T_{sm} = \mathbf{d}'\mathbf{V}^{-1}\mathbf{d}$$

where $\mathbf{d}$ is a column vector with elements equal to the differences $d_i = n_{i.} - n_{.i}$ for $i = 1, 2, \ldots, K$ and $\mathbf{V}$ is the variance-covariance matrix with elements:

$$v_{ii} = n_{i.} + n_{.i} - 2n_{ii}$$
$$v_{ij} = -(n_{ij} + n_{ji}), i \neq j$$

T_{sm} is asymptotically χ^2 with $(K-1)$ degrees of freedom.

The Stuart–Maxwell test statistic properly accounts for the dependence between the table's rows and columns, however, it requires the inversion of the non-diagonal matrix, $\mathbf{V}$. When the table is sparse the matrix may not be of full rank and, in that case, the commands substitute a generalized inverse $\mathbf{V}^*$ for $\mathbf{V}^{-1}$. An optional marginal homogeneity statistic that does not require the inversion of the variance–covariance matrix was suggested by Bickenböller and Clerget-Darpoux (1995) This test is available as an option to the symmetry command.

$$T_{mh} = \sum_{i} \frac{(n_{i.} - n_{.i})^2}{n_{i.} + n_{.i}}$$

This statistic is asymptotically distributed, under the assumption of marginal independence, as χ^2 with $(K-1)$ degrees of freedom.

This test statistic is reported when option mh is specified. This statistic, T_{mh}^o, is calculated in the same way as T_{mh} however the diagonal elements do not enter into the calculation of the marginal totals. Unlike the previous test statistic, this one reduces to a McNemar for 2×2 tables. The test statistic $[(K-1)/2]T_{mh}^o$ is asymptotically distributed as χ^2 with $(K-1)$ degrees of freedom (Cleves et al. 1997, Spieldman and Ewens 1996).

Exact symmetry test

An exact test of symmetry is provided for use on sparse tables. This test is computationally intensive and thus should not be used on large tables. The test is based on a permutation algorithm applied to the null distribution. The distribution of the off diagonal elements n_{ij}, $i \neq j$ conditional on the sum of the complementary off-diagonal cells, $N_{ij} = n_{ij} + n_{ji}$, can be written as the product of $K(K-1)/2$ binomial random variables:

$$P(\mathbf{n}) = \prod_{i<j} \binom{N_{ij}}{n_{ij}} \pi_{ij}{}^{n_{ij}} (1 - \pi_{ij})^{n_{ij}}$$

where $\mathbf{n}$ is a vector with elements n_{ij} and $\pi_{ij} = E(n_{ij}/N_{ij}|N_{ij})$. Under the null hypothesis of complete symmetry, $\pi_{ij} = \pi_{ji} = 1/2$, and thus the permutation distribution is given by:

$$P(\mathbf{n}) = \prod_{i<j} \binom{N_{ij}}{n_{ij}} \left(\frac{1}{2}\right)^{N_{ij}}$$

The exact significance test is performed by evaluating:

$$P_{cs} = \sum_{n \in p} P_0(\mathbf{n})$$

where $p = \{n : P_0(\mathbf{n}) < P_0(\mathbf{n}^*)\}$ and $\mathbf{n}^*$ is the observed contingency table data vector. The algorithm evaluates p_{cs} exactly.

Example

Consider a survey of 344 individuals (BMDP 1990, 267–270). Each person was asked in October 1986 whether they agreed or disagreed with then President Reagan's handling of foreign affairs. In January 1987, after the "Iran-Contra" affair became public these same individuals were surveyed again and asked the same question. We would like to know if public opinion changed over this time period.

Lets first describe the data and list a few observations.

```
. describe
Contains data from iran.dta
  obs:           344
 vars:             2
 size:         2,064 (98.6% of memory free)
-------------------------------------------------------------------------
   1. before      byte   %8.0g
   2. after       byte   %8.0g
-------------------------------------------------------------------------
Sorted by:
.   list in 1/5
        before      after
  1.     agree      agree
  2.     agree   disagree
  3.     agree     unsure
  4.  disagree      agree
  5.  disagree   disagree
```

Each observation corresponds to one of the 344 individuals. The data is in wide form thus each observation has a before and an after measurement. We first perform the test for symmetry without options.

```
. symmetry before after

----------+------------------------------------
          |              after
   before |    agree   disagree    unsure     Total
----------+------------------------------------
    agree |      47         56         38       141
 disagree |      28         61         31       120
   unsure |      26         47         10        83
          |
    Total |     101        164         79       344
----------+------------------------------------


                          Chi-Squared    df    Prob>chi2
-----------------------+------------------------------
Symmetry               |    14.8654      3       0.0019
Marginal homogeneity (MH) |  14.7783     2       0.0006
-----------------------+------------------------------
```

The test first tabulates the data in a $K \times K$ table and then performs Bowker's test for table symmetry, and Stuart–Maxwell's test for marginal homogeneity. The same results would be generated if the option `notable` was specified, however the cross-tabulation table would not be produced.

Both the symmetry test and marginal homogeneity test are highly significant indicating a shift in the responders' perception. We can further examine the cells responsible for this significant result by specifying the `contrib` option. We will also specify the `exact` option because we are working on a fast computer and have sufficient memory to handle this size table.

```
. symmetry before after, contrib exact mh

----------+------------------------------------
          |              after
   before |    agree   disagree    unsure     Total
----------+------------------------------------
    agree |      47         56         38       141
 disagree |      28         61         31       120
   unsure |      26         47         10        83
          |
    Total |     101        164         79       344
----------+------------------------------------


                    Contribution
                     to Symmetry
      Cells          Chi-Square
--------------      --------------
n1_2 & n2_1            9.3333
n1_3 & n3_1            2.2500
n2_3 & n3_2            3.2821


                          Chi-Squared    df    Prob>chi2
-----------------------+------------------------------
Symmetry               |    14.8654      3       0.0019
Marginal homogeneity (MH) |  14.7783     2       0.0006
MH (Bickenboller)      |    13.5272      2       0.0012
MH (no diagonals)      |    15.2494      2       0.0005
-----------------------+------------------------------

Symmetry (exact significance probability)           0.0018
```

The largest contribution to the symmetry χ^2 is due to cells n_{12} and n_{21}. These correspond to changes between the agree and disagree categories. Of the 344 individuals 58 (16.3%) changed from the agree to the disagree response while only 28 (8.1%) changed in the opposite direction.

For these data, the result from the exact test is similar to that obtained from the asymptotic test.

Saved Results

`symmetry` saves results in `S_#` macros.

S_1	number of pairs
S_2	symmetry χ^2
S_3	symmetry df
S_4	symmetry p value
S_5	MH (Stuart-Maxwell) χ^2
S_6	MH (Stuart-Maxwell) df
S_7	MH (Stuart-Maxwell) p value
S_8	MH (Bickenböller) χ^2
S_9	MH (Bickenböller) df
S_10	MH (Bickenböller) p value
S_11	MH (no diagonals) χ^2
S_12	MH (no diagonals) df
S_13	MH (no diagonals) p value
S_14	Exact symmetry p value

References

Bickenböller, H. and F. Clerget-Darpoux. 1995. Statistical properties of the allelic and genotypic transmission/disequilibrium test for multiallelic markers. *Genetic Epidemiology* 12: 865–870.

BMDP. 1990. BMDP statistical software manual. Example 4F2.9. Los Angeles; BMDP Statistical Software, Inc.

Bowker A. H. 1948. A test for symmetry in contingency tables. *Journal of the American Statistical Association* 43: 572–574.

Cleves M. A. , J. M. Olson, and K. B. Jacobs. 1997. Exact transmission–disequilibrium tests with multiallelic markers. *Genetic Epidemiology* 14: 337–347.

Maxwell A. E. 1970. Comparing the classification of subjects by two independent judges. *British Journal of Psychiatry* 116: 651–655.

Spieldman R. S. and W. J. Ewens. 1996. The TDT and other family-based tests for linkage disequilibrium and association. *American Journal of Human Genetics* 59: 983–989.

Spieldman R. S. , R. E. McGinnis, and W. J. Ewens. 1993. Transmission test for linkage disequilibrium: The insulin gene region and insulin-dependents diabetes mellitus. *American Journal of Human Genetics* 52: 506–516.

Stuart A. 1995. A test for homogeneity of the marginal distribution in a two way classification. *Biometrika* 32: 412–416.

ssa10	Analysis of follow–up studies with Stata 5.0

David Clayton, MRC Biostatistical Research Unit, Cambridge, david.clayton@mrc-bsu.cam.ac.uk
Michael Hills, London School of Hygiene and Tropical Medicine (retired), mhills@regress.demon.co.uk

In epidemiology and demography, survival analysis methods are important for the analysis of event history data. However, these disciplines have special requirements which differ from those of clinical trial analysts, whose needs have dominated the design of software over the last 25 years. Principal amongst these are

- long follow-up times with time-varying covariates,

- late entry, or left truncation,

- possible involvement of more than one time scale (for example age and calendar time), and

- interest in more than one type of "failure" event.

In rationalizing its approach to survival time (st) data, Stata 5.0 went a long way to addressing some of these needs. This submission adds some further tools and suggestions.

In STB-27 we introduced three commands for analyzing follow-up studies using the simple tabulation and stratification methods described in Part I of Clayton and Hills (1993). These were called lexis, tabrate, and mhrate. We have converted our commands into the st form and added some new features. The new versions are called stlexis, strate and stmh. We have also added some new commands. A brief description of these commands follows:

stlexis expands a set of records for subjects in a follow-up study into a larger number of records. Each new record concerns the follow-up of one subject through one band of a time scale.

strate tabulates rates by one or more categorical variables. The summary dataset, including event counts and rate denominators, can be saved for further analysis or display. The combination of the commands stlexis and strate implements all the functions of the special purpose "person-years" programs in widespread use in epidemiology.

stmh calculates stratified rate ratios and significance tests using a Mantel–Haenszel-type method.

staalen plots the cumulative rate (integrated hazard) against time. It can also be used after Cox regression to plot the cumulative *baseline* rate.

stmc calculates rate ratios stratified finely by time, using the Mantel–Cox method. The corresponding significance test (the
   log-rank test) is also calculated.

sttocc creates a nested case-control study from a follow-up study by sampling risk sets.

sttody converts st data to dy (event-count and person-time) data to allow, for example, analysis by Poisson regression.

dyrate is an extended and renamed version of the old tabrate.

dymh is an extended and renamed version of the old mhrate.

   In a follow-up study the date of entry and the date of exit are usually recorded as calendar dates, which are converted to
days since 1/1/1960 in Stata. When several time scales are relevant during the course of an analysis, it is natural to keep this as
the basic time scale and to map onto other scales with origin and scale options (see below). This often leads to negative times.
Negative times are also involved when time is measured from some important event such as heart transplantation; follow-up
before transplantation then takes place in negative time. The current version of stset does not allow negative times, so if you
have dates before 1/1/1960 one easy solution is to add 36525 days to all your dates, and carry out the analysis in the 21st
Century. Stata Corporation is aware of this problem and plans to change all st commands to accept negative times in the next
release.

   Many of our new st commands share the options origin and scale which control the mapping from the time scale
declared in stset (which we shall call the *basic* scale) onto the time scale on which the analysis is to be performed (the
analysis scale). These options avoid the need to constantly recall stset to redefine the date on a different scale. Time scales
differ only in their origin; to switch to age as the time scale the origin must be set at the date of birth for each subject; to
switch to time-since-entry as the time scale the origin is set to date of entry, and so on. The origin option declares a variable
(or constant) which specifies the time origin for each record. The scale option makes it possible to specify new *units* for time;
if the basic units are days declaring the scale to be 365.25 will specify the analytical units to be years. The origin and units of
the basic time variables in stset do not change.

   The Stata command stset requires a failure variable which indicates the outcome at the end of follow-up. This must be 0
for follow-up which is censored, but if the follow-up ends with an event, then the failure variable can be coded to indicate the
type of event. The convention in Stata 5.0 is to treat all non-zero codes as failures, which means that when analyzing a number
of different outcomes it is necessary to re-define the outcome variable using stset for each. We have avoided the need for this
in our st commands by introducing an fcodes (failure codes) option which specifies those codes which are to be regarded as
failures, all others to be treated as censored. The default is to include all non-zero codes as failures. Again our motivation has
been to avoid the need for repeated calls to stset. We hope that this submission will provide a convincing case for routinely
incorporating fcodes, origin, and scale options in st commands as appropriate.

   Each individual record in st data refers to a period of follow-up (its start and end) and the failure code indicating the nature
of the terminating event. Other commands, also useful in the analysis of event history data, allow for records describing the
occurrence of more than one event and expect the data in a somewhat different form, each record containing an event count and
a corresponding rate denominator. Examples of such commands are poisson, nbreg, and xtpois. We call such data dy data,
deriving from the way we name the event count and rate denominator variables in our book, and provide a conversion program,
sttody, which converts st data to dy data. Similar data arises in the person-years method of analysis of epidemiological cohort
studies in which counts of incident events and corresponding person-years observations are counted in cells of a multiway table
and subsequently analyzed using Poisson regression. To facilitate such analyses we have included in strate the facility to save
such multiway tables as Stata .dta files.

Subdivision of follow-up time by bands using stlexis

   The command stlexis expands a set of records for subjects in a follow-up study into a larger number of records. Each
record in the new dataset concerns the follow-up of one subject through one band of a time scale. Expansion by several time
scales can be achieved by repeated calls to stlexis. The id variable must be defined using stset in order to relate the new
records back to the subject to whom they refer. Since the current dataset will be altered by this command, any if and in options
are best implemented using keep or drop.

   The syntax of stlexis is

   stlexis $\left[varlist\right]$, <u>br</u>eaks($x_1, x_2, \ldots, x_k$) $\left[\right.$ <u>or</u>igin(*varname* | #) <u>sc</u>ale(*varname* | #)

      <u>gen</u>erate($\left[type\right]$ *varname*) <u>l</u>abel <u>i</u>nvert $\left.\right]$

The *varlist* specifies which variables (in addition to the `st` data) should be kept in the expanded dataset. If it is absent, all variables are kept. The `st` data definition remains in force after the expansion and the `st` variables are correctly recoded to refer to the new records.

Options

`breaks`$(x_1, x_2, \ldots, x_k)$ is not optional. It supplies the breaks for the bands, in ascending order, in the units of (time−origin)/scale where time has the units specified in `stset`. The list of break points may be simply a list of numbers separated by commas, but may also include the syntax a[b]c, meaning from a to c in steps of size b. Thus

```
40, 45, 50, 55, 60, 70, 80, 90

40[5]60,70[10]90

40[5]60[10]90
```

are all valid and describe the same list.

`origin`(*varname* | #) specifies the origin of the analysis time scale. The default is 0.

`scale`(*varname* | #) specifies the units for the analysis time scale. The default is 1.

`generate`([*type*] *varname*) supplies the name of a new variable to hold the time band indicators, coded using the left-hand ends of the bands defined by `breaks`. If it is not defined, the default variable `t_band` is used.

`label` controls the coding and labeling of the generated time band variable. By default the variable is coded to the lower break point for the time band and is unlabeled. If `label` is set, bands are coded 0, 1, 2 . . . , and appropriate labels are computed.

`invert` reverses the roles of the basic and analysis scales in defining the mapping by origin and scale.

Comments

The first and last break points define the span of the study, according to the following rules. Records for which the time of exit from the study is less than the first break point, and records for which the time of entry is greater than the last break point, are dropped. Otherwise, the time of entry is redefined as the larger of time at entry and the first break point, and the time of exit is redefined as the smaller of time at exit and the largest break. For records in which the time of exit is greater than the largest break point the failure indicator is set to zero (censored), no matter what its original value was.

The command may be repeated with different time scales, thus partitioning observations between cells in the *Lexis diagram*. This explains its name.

For advanced users, `scale` can be a variable, allowing different units for different records. This is useful for computations based on cumulative dose of environmental exposure.

We shall illustrate the use of `stlexis` with the same data as in STB-27 but the time variables are now given as dates, and the outcome variable has been left fully coded, with codes 1, 3, and 13 referring to different types of coronary heart disease (CHD) event. Other events, such as cancer incidence, are coded with different positive integers and censored follow-up (no event at end of study) is coded 0.

```
1. id        float  %9.0g              Subject identity number
2. doe       long   %dDmCY             Date of entry
3. dox       long   %dDmCY             Date of exit
4. dob       long   %dDmCY             Date of birth
5. fail      int    %8.0g              Outcome (CHD = 1 3 13)
6. job       int    %8.0g              Occupation
7. month     byte   %8.0g              month of survey
8. energy    float  %9.0g              Total energy (Mcals/day)
9. hieng     byte   %8.0g              Indicator for high energy
```

The variable `hieng` is coded 1 if the total energy consumption is > 2.75 Mcals and 0 otherwise.

Example 1: Age-specific rates

We shall first expand the data using age as the time scale with 10-year age bands. Note that the origin is set to date of birth, making age the time scale, and the scale is set to 365.25, so that the breaks can be specified in years of age.

```
. use diet
. stset dox fail, t0(doe) id(id)
```

```
. stlexis, gen(ageband) br(40,50,60,70) origin(dob) scale(365.25)
Lexis: expansion of survival time data by time bands
    failure time:  dox
      entry time:  doe
  failure/censor:  fail
              id:  id

          origin:  dob
         scaling:  365.25
Time band will be coded in the variable: ageband
26   records start before 40 - left truncated
392  extra records are being created
```

The effect of the `stlexis` command on the data is shown by

```
. list id doe dox ageband fail if id==1
            id        doe         dox      ageband       fail
    1.       1   16Aug2064   03Jan2065         40          0
    2.       1   03Jan2065   04Jan2075         50          0
    3.       1   04Jan2075   01Dec2076         60          0
. list id doe dox ageband fail if id==34
            id        doe         dox      ageband       fail
   61.      34   16Apr2059   12Jun2059         50          0
   62.      34   12Jun2059   31Dec2066         60          3
```

This shows how the single record for subject with id = 1 has expanded to three records. The first refers to the age band 40–49, coded 40, and the subject spends from 16Aug2064 to 03Jan2065 in this band. The second refers to the age band 50–59, coded 50, and the subject spends from 03Jan2065 to 04Jan2075 in this band, and so on. The follow-up in each of the three bands is censored (fail = 0). The single record for the subject with id = 34 is expanded to two age bands; the follow-up for the first band was censored (fail = 0) and the follow-up for the second band ended in CHD (fail = 3).

The values for variables which do not change with time, such as *height*, are simply repeated in the new records. This can lead to much larger datasets after expansion, and it may be necessary to specify which variables are to be kept after expansion, using the option to do this in `stlexis`.

Example 2: Age and time-in-study

To use `stlexis` to expand the records on two time scales, such as age and time-in-study, first expand on the age scale and then on the time-in-study scale.

```
. use diet,clear
. stset dox fail, t0(doe) id(id)
. stlexis, gen(ageband) br(40[10]70) origin(dob) scale(365.25)
. stlexis, gen(timeband) br(0[5]25) origin(doe) scale(365.25)
. list id doe dox ageband timeband fail if id==1
            id        doe         dox      ageband    timeband      fail
    1.       1   16Aug2064   03Jan2065         40          0          0
    2.       1   03Jan2065   03Jan2070         50          0          0
    3.       1   03Jan2070   04Jan2075         50          5          0
    4.       1   04Jan2075   01Dec2076         60          0          0
. list id doe dox ageband timeband fail if id==34
            id        doe         dox      ageband    timeband      fail
   90.      34   16Apr2059   12Jun2059         50          0          0
   91.      34   12Jun2059   11Jun2064         60          0          0
   92.      34   11Jun2064   31Dec2066         60          5          3
```

Example 3: Explanatory variables which change with time

In the previous examples time itself, in the shape of age or time in study, is the explanatory variable which is to be studied or controlled for, but in some studies there are other explanatory variables which vary with time. The `stlexis` command can sometimes be used to expand the records so that in each new record such an explanatory variable is constant over time. For example, in the Stanford heart data (see [R] **st stset**) the explanatory variable is `posttran` which takes the value 0 before transplantation and 1 after. The follow-up must therefore be divided into time before transplantation and time after. This can be achieved with

```
. stlexis, gen(posttran) br(-10000,0,10000) origin(dtrans) label
. label drop posttran
```

where `dtrans` contains the date of transplant, set equal to (say) 1/1/2000 for subjects still waiting for a transplant. The command breaks the follow-up into the interval from -10000 to 0 and from 0 to 10000. Because the `label` option is in force, the bands are coded 0 and 1 respectively in `posttran`. These bands correspond to time before and after the transplant. The actual labels for the codes 0, 1 are not very helpful here, and have been dropped.

Example 4: Breaking down follow-up by cumulative exposure

Cumulative exposure to environmental pollutants can behave in very much the same way as another time scale. Here, however, the rate of passage of "time" varies from one record to another as a result of different levels of exposure. For example, in radiation studies in which measurements of cumulative exposure to ionizing radiation are available, the rates are worked out using follow-up time during which the cumulative exposure is $[0, 100)$, follow-up time during which the cumulative exposure $[100, 200)$, etc. This can be done using `stlexis` by defining cumulative exposure as the analysis "time" scale and specifying the breaks in terms of the cumulative exposure. As an example, consider the following made-up data which are in the file `cumdose`.

```
id timein timeout xin xout fail
1  0      5         0    40  0
1  5      10        40  180  0
1 10      15       180  220  1
```

The three records all refer to subject 1, the basic time scale is time since entry to the study (years), and x refers to cumulative exposure. Thus the subject enters at time 0 with zero exposure; after 5 years this has gone up to 40; after 10 years to 180, and after 15 years to 220 at which time the subject fails. The idea is to break the follow-up time into parts which correspond to the cumulative exposure groups defined by the cut points (0, 100, 200, 300).

The rate at which exposure is accumulated is

Follow-up in years	Exposure per year
0–5	$(40 - 0)/(5 - 0) = 8$
5–10	$(180 - 40)/(10 - 5) = 28$
10–15	$(220 - 180)/(15 - 10) = 8$

The first 5 years of follow-up belong entirely to the exposure group $[0, 100)$. Assuming a linear increase in exposure between readings it would take 2.143 years at a rate of 28/year to get the exposure from 40 to 100, and a further 2.857 years would bring the exposure to 180, so the second record should be broken into a piece lasting 2.143 years which belongs to $[0, 100)$, and a piece lasting 2.857 years which belongs to $[100, 200)$. In the final record it would take 2.5 years at 8/year to bring the exposure to 200, and a further 2.5 years brings it to 220 so the final record should be split into a piece lasting 2.5 years which belongs to $[100, 200)$ and a piece lasting 2.5 years which belongs to $[200, 300)$.

If we used `stlexis` in the same way as before, we would need to set the `scale` to be a variable equal to the *reciprocals* of the values shown in the last column of the above table—that is, as years per unit of exposure. Unfortunately this does not allow for periods in which a subject is unexposed and we have had to create an `invert` option which reverses the roles of basic and analytical time scales in the definition of origin and scale. With `invert` in force, `scale` sets the number of units of the analytical time scale corresponding to one unit of the basic time scale, and `origin` declares the point on the analysis time scale corresponding to zero on the basic time scale. These calculations are illustrated below.

```
. use cumdose, clear
. stset timeout fail, t0(timein) id(id)
. gen scale = (xout-xin)/(timeout-timein)
. gen origin = xin - timein*scale
. stlexis, origin(origin) scale(scale) invert br(0[100]300) gen(egrp)
```

```
            Lexis: expansion of survival time data by time bands
                failure time:  timeout
                  entry time:  timein
             failure/censor:  fail
                          id:  id
                      origin:  origin
                               (value, on break-scale, when t=0 on st-scale)
                     scaling:  scale
                               (units on break-scale per unit on st-scale)
        Time band will be coded in the variable: egrp
        2    extra records are being created
```

To check that the records have been expanded correctly:

```
    . list id timein timeout xin xout egrp
                id      timein      timeout         xin        xout        egrp
        1.       1           0            5           0          40           0
        2.       1           5     7.142857          40         180           0
        3.       1    7.142857          10          40         180         100
        4.       1          10         12.5         180         220         100
        5.       1        12.5           15         180         220         200
```

Tabulating the rate

The st version of tabrate is called strate and has syntax

strate [*varlist*] [in *range*] [if *exp*] [using *filename*] [, <u>sc</u>ale(#) <u>fc</u>odes(*codes*) <u>j</u>ack

<u>cl</u>uster(*varname*) smr(*varname*) <u>nol</u>ist <u>nom</u>iss <u>rep</u>lace <u>now</u>hisker <u>l</u>evel(#) <u>g</u>raph *graph_options*]

strate calls dyrate and tabulates the rate, formed from the number of failures divided by the person-time, by different levels of one or more categorical explanatory variables declared in the *varlist* of the command. Confidence intervals for the rate are also given. By default, confidence intervals for rates are calculated using the quadratic approximation to the Poisson log-likelihood for the log rate parameter. However, for situations in which the Poisson assumption is questionable, jackknife confidence intervals can also be calculated. The jackknife option also allows for the case where there are multiple records for the same cluster (usually subject).

The command also implements computation of SMR's, after merging the data with a suitable file of reference rates.

The summary dataset can be saved to a file (specified with the using clause), thus enabling further analysis or more elaborate graphical display.

Weights may be specified with the stset command. If they are used, the program calculates jackknife confidence intervals by default.

Options

scale(#) specifies the units for the analysis time scale.

fcodes(*codes*) specifies the codes for the failure indicator to be treated as failures in the analysis. All other codes are treated as censoring. Abbreviations such as fcodes(1/3 5/7 9) are allowed (as in recode).

jack specifies that jackknife confidence intervals are required.

cluster(*varname*) defines a categorical variable which indicates clusters of data to be used by the jackknife. If the jackknife option is selected and this variable is not declared, it is taken as the id variable defined in the st data.

smr(*varname*) specifies a variable which holds, for each record, an appropriate reference rate. The program then calculates SMR's rather than rates. This option will usually be used after using stlexis to split the follow-up records by age bands (and possibly calendar periods).

nolist suppresses the output. This is only useful when saving the results to a file with using.

nomiss restricts analysis to records with no missing values in any of the explanatory variables. Otherwise missing values are simply treated as extra categories.

replace allows overwriting of the file specified with using

nowhisker omits the confidence intervals from the graph.

`level(#)` resets the level for the confidence intervals from the usual default.

`graph` produces a graph of the rate against the numerical code used for the categories of *varname*. Graph options are allowed.

Example 5: Tabulating the age-specific CHD rates

After expanding on the age scale using breaks 40[10]70 (Example 1), the dataset consists of 729 records. The CHD rate per 1000 person-years can now be tabulated against ageband using the command

```
. strate ageband, fc(1 3 13) scale(365250)

 ageband   _D        _Y    _Rate   _Lower   _Upper
      40    6   0.90701    6.615    2.972   14.725
      50   18   2.10703    8.543    5.382   13.559
      60   22   1.49330   14.732    9.701   22.374
```

A note on standardized mortality ratios (SMRs)

The SMR for a cohort is the ratio of the total number of observed deaths to the number expected from age-specific reference rates. This expected number can be found by first expanding on age, using `stlexis`, and then multiplying the person years in each age band by the reference rate for that band. The Stata command `merge` can be used after expansion by `stlexis` to add the reference rates to the dataset and, using the `smr` option to define the variable containing the reference rates, `strate` can be used to calculate SMR's and confidence intervals. Note that the `scale` must still be set—if reference rates are per 100,000 then the scale should be 36525000. When reference rates are available by age and calendar period, `stlexis` must be called twice to expand on both time scales before merging the data with the reference rate file.

Plotting cumulative rates

We have seen above how the variation of rates along a time scale can be studied by first breaking up each subject's follow-up into parts and then calculating time-band-specific rates. A disadvantage of this in practice is that the break points may be arbitrary and dividing into too narrow bands results in unstable estimates of rates. An alternative procedure is to divide time into very short bands (referred to as "clicks" in Clayton and Hills), and plotting the *cumulative rate* against time. In survival analysis the cumulative rate is often referred to as the *integrated hazard*. Since several such plots can be superimposed, this technique is very useful for making informal comparisons between time-specific rates in different groups. Such plots can be produced using `staalen`, which has syntax:

staalen [in *range*] [if *exp*] [, <u>or</u>igin(*varname* | #) <u>sc</u>ale(#) <u>fc</u>odes(*codes*) by(*varname*)

<u>rm</u>ult(*varname*) <u>fr</u>om(#) to(#) risk *graph_options*]

The cumulative rate is obtained by cumulating $1/N$, where N is the number at risk just before each event. This estimate is usually known as the Aalen–Nelson estimate of cumulative rate. A plot of the cumulative rate against time has a positive step at the time of each event.

The command `staalen` plots the Nelson–Aalen estimate of the cumulative rate or intensity. When the `rmult()` option contains the rate multipliers for each record, predicted by a Poisson or Cox regression, the estimate becomes the Breslow–Aalen estimate of the cumulative baseline rate in the proportional hazards model.

The cumulative risk can be plotted instead of the cumulative rate, using the identity

$$\text{Cumulative risk} = 1 - \exp(-\text{Cumulative rate})$$

Note that there are difficulties in interpreting the cumulative risk when there are competing causes of failure.

Options

`origin(`*varname* | #`)` specifies the origin for the time scale. The default is 0.

`scale(#)` specifies the units for time scale. The default is 1.

`fcodes(`*codes*`)` specifies those codes to be included as failures.

`by(`*varname*`)` specifies a categorical variable by which the cumulative rate will be plotted.

`rmult(`*varname*`)` contains the rate multipliers for each record obtained from fitting a proportional hazards model, for example, using `stcox`.

from(#) provides left truncation of all observations at a fixed lower time limit. The from and to options provide the ability to "zoom in" on one part of the time scale.

to(#) provides right censoring of all observations at a fixed upper time limit.

risk causes the cumulative risk rather than the cumulative rate to be estimated

graph_options are allowed. Default labeling is supplied when *graph_options* are absent, but the x-axis may be relabeled using the b2 graphics option and the y-axis may be relabeled using the l1 option.

Example 6: Plotting the cumulative rate

To plot the cumulative rate for the diet data, using age as the time scale, separately by hieng, use

```
. use diet
. stset dox fail, t0(doe) id(id)
. staalen, origin(dob) scale(365.25) fc(1 3 13) by(hieng)
```

The graphs are shown in Figure 1. In both groups the rate is roughly constant with age after about 45 (the plots are linear) but the rate for the high energy group is lower than for the other group. Note how the time scale is labeled; this could be relabeled by adding l1(Cumulative rate by hieng) b2(Age) to the command.
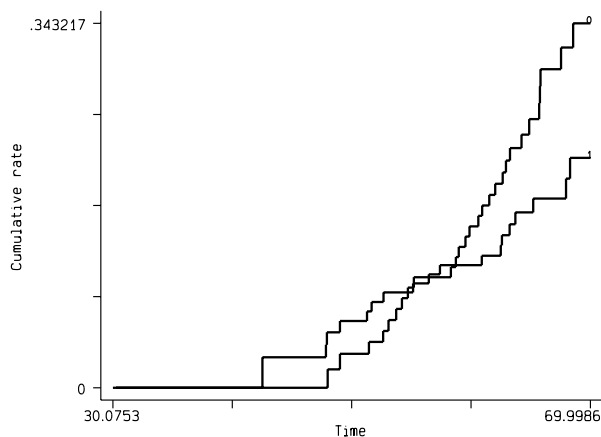

Figure 1

Rate ratios and log-linear trend

The command stmh is used for estimating rate ratios, controlled for confounding using stratification, and has syntax

stmh *varname* [*varlist*] [in *range*] [if *exp*] [, <u>fc</u>odes(*codes*) <u>c</u>ompare(*codes1*, *codes2*)

by(*varlist*) <u>l</u>evel(#) <u>nom</u>iss]

The command stmh calls dymh, and in its simplest use, estimates the ratio of the rates of failure for two categories of the explanatory variable (the first argument). Categories to be compared may be defined by specifying the codes of the levels to be compared, for example as c(1/3 8, 5/7 9). Alternatively the command may be used to carry out trend tests for a metric explanatory variable. In this latter case a one-step Newton approximation to the log-linear Poisson regression coefficient is also computed.

The remaining variables before the comma are categorical variables which are to be controlled for using stratification. Strata are defined by cross-classification by all of these variables and the rate ratio estimate is combined over strata using the Mantel–Haenszel method. Using the by option, the variation of the rate ratio with further categorical variables may be explored. By default, missing values of these variables define new strata, but there are arguments against this and an alternative behavior—omitting any records with missing values for these variables—may be selected using the nomiss option.

On completion, the macro S_1 contains the overall Mantel–Haenszel estimate of the rate ratio, thus enabling bootstrap evaluation of confidence intervals.

Options

fcodes(*codes*) specifies a recode rule for the failure indicator in the st data. All codes matching the rule are treated as failures and all others as censoring.

compare(*codes1*,*codes2*) specifies recode rules which define the categories of the exposure variable to be compared. The first rule defines the numerator categories and the second the denominator categories. When compare is absent and there are only two categories, the larger is compared to the smaller; when there are more than two categories an analysis for (log-linear) trend is carried out.

by(*varlist*) specifies categorical variables by which the rate ratio is to be tabulated. A separate rate ratio produced for each category or combination of categories, and a test for unequal rate ratios (effect modification) is given. In an analysis for log-linear trend, this test is very approximate since the estimates are themselves based on a quadratic approximation to the log-likelihood.

level(*#*) gives the level for the confidence intervals (default 95).

nomiss specifies that only cases that have no missing values for any stratifying variables should be included. By default missing values will define new strata.

Example 7: Stratified rate ratios

After expanding the records using stlexis on the age scale, the rate ratio for hieng, level 1 compared to level 0, controlled for ageband, can be found using

```
. stmh hieng, c(1,0) fc(1 3 13) by(ageband)
Maximum likelihood estimate of the rate ratio
  comparing: hieng==1 vs hieng==0
  by: ageband
RR estimate, and lower and upper 95% confidence limits
    ageband      RR    Lower    Upper
         40    1.239    0.227    6.762
         50    0.434    0.163    1.157
         60    0.503    0.211    1.199
Overall estimate controlling for:  ageband
     RR    Lower    Upper    Chisq    p_value
  0.534    0.293    0.972    4.357    0.03686
Approx chisq for unequal RRs (effect modification)    1.19 (1 df, p = 0.27520)
```

Note that since the RR estimates are approximate, the test for unequal rate ratios is also approximate.

Example 8: Effect modification

The effect of hieng, controlled for ageband, can be compared between jobs with

```
. stmh hieng ageband, fc(1 3 13) c(1,0) by(job)
Mantel--Haenszel estimate of the rate ratio
  comparing: hieng==1 vs hieng==0
  controlling for: ageband
  by: job
RR estimate, and lower and upper 95% confidence limits
      job      RR    Lower    Upper
        0    0.418    0.131    1.334
        1    0.639    0.218    1.873
        2    0.514    0.210    1.256
Overall estimate controlling for: ageband job
     RR    Lower    Upper    Chisq    p_value
  0.521    0.289    0.939    4.883    0.02713
Approx chisq for unequal RRs (effect modification)    0.28 (1 df, p = 0.59685)
```

Example 9: Metric explanatory variables

This example illustrates what happens when *varname* refers to a metric variable, in this case *height*. The first command tests for a trend of heart disease rates with height within age bands, and also provides a rough estimate of the rate ratio for a 1 cm increase in height—this estimate is a one-step Newton approximation to the maximum-likelihood estimate, and is not consistent, but it does provide a useful indication of the size of the effect.

```
. stmh height, fc(1 3 13)
Score test for trend of rates with: height
  with an approximate estimate of the
  rate ratio for one unit increase in height
RR estimate, and lower and upper 95% confidence limits
    RR    Lower    Upper    Chisq   p_value
  0.907    0.868    0.949   18.247   0.00002
```

There is clear evidence for a decreasing rate with increasing height (about 9% decrease in rate per cm of height).

Rate ratios stratified finely by time

To use `stmh` to control for variation of rates along a time scale, it is first necessary to use `stlexis` to break the follow-up into parts corresponding to different bands of time. Our next command combines these steps, breaking up time into very short intervals, or *clicks*. Usually this approach is only used to calculate significance tests and the resultant test has been christened the Mantel–Cox or log-rank test. However, the rate ratio estimates remain just as useful as in the coarsely stratified analysis described above. The method may be viewed as an approximate form of Cox regression. The name of the command is `stmc`, from Mantel–Cox, and it has syntax

stmc *varname* [*varlist*] [if *exp*] [in *range*] [, origin(*varname*) fcodes(*codes*) compare(*codes1*,*codes2*)

      by(*varlist*) nomiss level(*#*)]

The rate ratio produced will be controlled for time, with origin as specified in `origin()`, separately for the different levels of the variables in the `by()`, and finally combined to give a rate ratio controlled for both time and the variables in the `by()`.

Options

The meaning and use of the various options should be clear from their use in `stlexis` and `stmh`. As with `stmh` the macro S_1 contains the overall estimate of the rate ratio.

Example 10: Controlling for age with fine strata

To obtain the effect of high energy controlled for age by stratifying very finely, use

```
. use diet, clear
. stset dox fail, t0(doe)
. stmc hieng, origin(dob) fc(1 3 13)
Mantel-Cox comparisons
      failure time:  dox
        entry time:  doe
    failure/censor:  fail
       time origin:  dob
      failure codes:  1 3 13
Mantel-Cox estimates of the rate ratio
  comparing: hieng==1 vs hieng==0
  controlling for: time (by clicks)
Overall Mantel-Cox estimate, controlling for: time-from-dob
   _RR  _Lower  _Upper  _Chisq  _p_value
  0.537   0.293   0.982   4.203   0.04035
```

The rate ratio of 0.537 is quite close to that obtained when controlling for age using 10-year age bands, namely

```
    RR    Lower    Upper    Chisq   p_value
  0.521    0.289    0.939    4.883   0.02713
```

Converting from st data to count data

In all `st` commands the failure variable indicates which periods of follow-up end in failure and which are censored, but each record can only refer to a period of follow-up and a single terminating event. Event histories with multiple events must be represented by multiple records sharing the same `id`—one record for each event, including the final end of follow-up.

However, for techniques such as Poisson regression (`poisson`), negative binomial regression (`nbreg`), and GEE for count data (`xtpois`), each record must describe an event count, together with an appropriate *rate denominator*. In our terminology, they require `dy` data. Such data are also required for our STB-27 commands `tabrate` and `mhrate` and we provide updated

versions of these, now renamed `dyrate` and `dymh` which implement all the new features of the corresponding `st` commands. They are not documented separately; the d variable is passed as an additional (first) argument and the y variable is specified, as in `poisson`, with an `e()` option. Further options are as for the equivalent `st` command although, of course, `fcodes`, `origin`, and `scale` options are no longer necessary.

To convert from `st` to `dy` form, we provide the command `sttody`. Its syntax is

> `sttody` *varname varname* $\big[$, <u>sc</u>`ale(`#`)` <u>fc</u>`odes(`*codes*`)` `by(`*varlist*`)` $\big]$

The arguments are new variables to contain, respectively, the event count, d, and the rate denominator, y. If there is no `by` option, one record is created for each unique value of the `id` variable declared in `stset`. Thus, when there are multiple records with the same `id`, the size of the dataset is reduced. Any time-varying variables will take on their first recorded value in the reduced dataset. The `by` option limits this collapsing down of multiple records into groups specified by a list of categorical variables. Thus, for example, all records for the same subject and which refer to the same age band could be grouped together and form the new record, by using a `by(`*ageband*`)` option. Thus, we can use `stlexis` followed by `sttody` to prepare recurrent event data for analysis using `xtgee`.

If no `id` variable is set, each record is taken as referring to a unique subject.

Options

`scale(`#`)` specifies the units for the analysis time scale, that is, units for y, the rate denominator.

`fcodes(`*codes*`)` specifies a recode rule for the failure indicator. All codes matching the rule are treated as failures and all others as censoring.

`by(`*varlist*`)` specifies the variables by which the data will be disaggregated within subject `id`.

Example 11: Using poisson regression with st data

After expanding the diet data on the age scale we used `stmh` to find the effect of high energy controlled for age. An alternative is to use Poisson regression. This is useful with metric variables and for controlling for a number of different variables. Poisson regression requires each record to contain a number of events (which can be zero) and a follow-up time, and `sttody` can be used to convert the diet data to this form.

```
. use diet, clear
. stset dox fail, t0(doe) id(id)
. stlexis , origin(dob) scale(365.25) breaks(40[10]70) gen(ageband)
. sttody d y, fcodes(1 3 13) scale(365.25) by(ageband)
```

The command

```
. poisson d hieng, e(y)
```

can now be used to obtain the effect of high energy.

Nested case-control studies

Any cohort study can be used to generate a case-control study by sampling the cohort for controls. The resultant case-control study is then said to be *nested* in the cohort study. For each case the controls are chosen from those members of the cohort who are at risk at the failure time of the case, that is from the *risk set* corresponding to the case. The case-control study so formed is matched with respect to the time scale used to compute risk sets, and when analyzing as a matched case-control study, odds ratios in the case-control study will estimate corresponding rate ratio parameters in the proportional hazards model for the cohort study.

Nested case-control studies are an attractive alternative to full Cox regression analysis, particularly when time-varying explanatory variables are involved. They are also attractive when some explanatory variables involve laborious coding; we create a file with a subset of variables for all subjects in the cohort, generate a nested case-control study, and go on to code the remaining data only for those subjects selected for the nested study.

In the same way as for Cox regression, the results of the analysis are critically dependent on the choice of time scale. The choice of time scale may be calendar time, so that controls would be chosen from subjects still being followed on the date that the case fails but other time scales, such as age or time-in-study, may be more appropriate in some studies. Remember that the scale used in selecting controls is implicitly included in the model in subsequent analysis. When drawing controls, they may also be matched to the case in respect to additional (categorical) variables such as sex. This produces an analysis closely mirroring a stratified Cox regression analysis. Note that we can carry out control selection after a call to `stlexis` so as to create, for

example, a case-control study by sampling from risk sets created in calendar time but also matched by 5- or 10-year age bands. Analysis as a matched case-control study estimates rate ratios in the underlying cohort which are controlled for calendar time (very finely) and age (less finely). Such analysis can be carried out by Mantel–Haenszel (odds ratio) calculations, for example using mhodds (STB-27), or by conditional logistic regression using clogit.

The command sttocc (survival time to case-control) can be used to generate a case-control study from the data from a cohort study. It has the syntax

sttocc [*varlist*] [, origin(*varname*) scale(*#*) fcodes(*codes*) match(*varlist*) number(*#*) generate(*varlist*)]

varlist defines variables which, in addition to those used in the creation of the case-control study, will be carried over. The default is that all variables are carried over into the case-control study.

Options

origin(*varname*) specifies the origin for the time scale.

scale(*#*) specifies the units for time scale.

fcodes(*codes*) specifies those codes to be included as failures.

match(*varlist*) specifies additional categorical variables for matching controls to cases.

number(*#*) specifies the number of controls to draw for each case. The default is 1, even though this is not a very sensible choice!

generate(*varlist*) specifies variable names for three generated variables. These are (with their default names) (i) a case-control indicator coded 0 for controls and 1 for cases (_case), (ii) a case-control set identifier (_set), and (iii) the time, on the analysis scale at which the set was constructed, that is the failure time of the case (_time).

Remark: treatment of ties

In the event of ties between entry times, censoring times, and failure times, the following convention is adopted:

$$\text{Entry time} < \text{Failure time} < \text{Censoring time}$$

Tied failure times are broken at random.

Example 12: Creating a nested case-control study

We shall illustrate the use of sttocc with the diet data, choosing age as the time scale, so that controls are chosen from subjects still being followed at the age at which the case fails. The commands

```
. use diet, clear
. stset dox fail, t0(doe) id(id)
. sttocc, origin(dob) scale(365.25) match(job) n(5)
```

create a new dataset in which there are 5 controls per case, matched on job, with the age of the subjects when the case failed recorded in the variable _time. The case indicator is given in _case and the matched set number in _set. All variables are carried over into the new dataset.

The commands

```
. gen ageentry=(doe-dob)/365.25
. gen ageexit=(dox-dob)/365.25
. list id _* age* in 1/12

              id      _case       _set       _time     ageentry      ageexit
    1.        93          0          1     42.57358     36.43258     51.32101
    2.        73          0          1     42.57358     36.58043     52.70636
    3.        75          0          1     42.57358     31.13484     47.26078
    4.       101          0          1     42.57358     38.29706      49.1718
    5.        65          0          1     42.57358     40.11225     56.82409
    6.        90          1          1     42.57358      31.4141     42.57358
    7.       190          0          2      47.8987      44.4846     64.52567
    8.       292          0          2      47.8987     46.24504     62.28611
    9.       285          0          2      47.8987     40.59138     57.30322
   10.       213          0          2      47.8987     47.23614     67.02532
   11.       305          0          2      47.8987      46.5462     48.04107
   12.       196          1          2      47.8987     45.46475      47.8987
```

demonstrate that controls did indeed belong to the appropriate risk set. Note that the controls in each set enter at an age which is less than that of the case at failure, and exit at an age which is greater than the age of the case at failure. To estimate the effect of high energy use `clogit`, just as you would for any matched case-control study:

```
. clogit _case hieng, group(_set) or

Iteration 0:  Log Likelihood =-143.26306
Iteration 1:  Log Likelihood =-143.04668
Iteration 2:  Log Likelihood =-143.04668

Conditional (fixed-effects) logistic regression       Number of obs =    480
                                                      chi2(1)       =   0.59
                                                      Prob > chi2   = 0.4431
Log Likelihood = -143.04668                           Pseudo R2     = 0.0021
------------------------------------------------------------------------------
   _case | Odds Ratio  Std. Err.       z    P>|z|      [95% Conf. Interval]
---------+--------------------------------------------------------------------
   hieng |  .8305188   .2011803    -0.767   0.443      .5166055     1.33518
------------------------------------------------------------------------------
```

References

Clayton, D. G. and M. Hills. 1993. *Statistical Models in Epidemiology*. Oxford: Oxford University Press.

——. 1995. ssa7: Analysis of follow-up studies. *Stata Technical Bulletin* 27: 19–26. Reprinted in *Stata Technical Bulletin Reprints*, vol. 5, pp. 219–227.

svy6	Versions of mlogit, ologit, and oprobit for survey data

John L. Eltinge, Texas A&M University, FAX 409-845-3144, jeltinge@stat.tamu.edu
William M. Sribney, Stata Corporation, FAX 409-696-4601, tech_support@stata.com

The syntax for the `svymlog`, `svyolog`, and `svyoprob` commands is

> svymlog *varlist* [*weight*] [if *exp*] [in *range*] [, <u>rrr</u> <u>b</u>asecategory(#) <u>noco</u>nstant
>
> *maximize_options svy_options*]
>
> svyolog *varlist* [*weight*] [if *exp*] [in *range*] [, *maximize_options svy_options*]
>
> svyoprob *varlist* [*weight*] [if *exp*] [in *range*] [, *maximize_options svy_options*]

where the *svy_options* are

> <u>str</u>ata(*varname*) psu(*varname*) fpc(*varname*) <u>sub</u>pop(*varname*) <u>srs</u>subpop <u>noadjust</u>
> <u>level</u>(#) <u>prob</u> ci deff deft meff meft

These commands share the features of all estimation commands. The commands typed without arguments redisplay previous results. The following options can be given when redisplaying results:

> rrr level(#) prob ci deff deft meff meft

`pweight`s are allowed. See [U] **18.1.6 weight** in the Stata User's Guide.

Warning: Use of `if` or `in` restrictions will not produce correct variance estimates for subpopulations in many cases. To compute estimates for a subpopulation, use the `subpop()` option.

[*Editor's note: The ado-files for these commands can be found in the stata directory.*]

Description

These commands are the equivalent of `mlogit`, `ologit`, and `oprobit` for complex survey data. The `svymlog` command estimates multinomial logistic regression; `svyolog` estimates ordered logistic regression; and `svyoprob` estimates an ordered probit model. Before using these commands, users should first familiarize themselves with the `mlogit`, `ologit`, `oprobit`, and `svyreg` commands.

The commands allow any or all of the following: probability sampling weights, stratification, and clustering. Associated variance estimates, design effects (`deff` and `deft`), and misspecification effects (`meff` and `meft`) are computed. The `subpop()`

option will give estimates for a single subpopulation defined by an expression; see [R] **svymean** and [R] **svyreg** in the Stata Reference Manual for a discussion of its properties.

To account for unequal selection probabilities, the customary maximum-likelihood estimating equations are weighted, with the specified pweights assumed to be proportional to the inverses of selection probabilities. Hence, these commands produce the same point estimates as those computed by mlogit, ologit, and oprobit when aweights are specified, but the variance estimator produced by svymlog, svyolog, and svyoprob is one that is appropriate for the survey design.

When these commands are used with nonsurvey data (i.e., no sampling weights, stratification, or clustering), they produce "robust" variance estimates—what nonsurvey statisticians term the Huber/White/sandwich estimator and what other commands in Stata give when the robust option is specified.

Options

The *svy_options* shown in the syntax diagram are the same options as those for the svyreg command; see [R] **svyreg** in the Stata Reference Manual for a description of those options.

rrr (svymlog only) reports the estimated coefficients transformed to relative risk ratios, i.e., $\exp(b)$ rather than b. Standard errors are also transformed to this metric. See [R] **mlogit** in the Stata Reference Manual for a description of relative risk ratios.

basecategory(#) (svymlog only) specifies the value of depvar that is to be treated as the base category. The default is to choose the most frequent category.

noconstant (svymlog only) estimates a model without the constant term (intercept).

maximize_options control the maximization process; see [R] **maximize** in the reference manual. You may want to specify the log option when estimating models on large datasets to view the progress of the maximum likelihood estimation steps. You should never have to specify the other *maximize_options*.

Examples

We use data from the Second National Health and Nutrition Examination Survey (NHANES II) (McDowell et al. 1981). We first set the strata, psu, and pweight variables:

```
. svyset pweight finalwgt
. svyset strata strata
. svyset psu psu
```

Once the strata, psu, and pweight variables are set, we can use svymlog, svyolog, or svyoprob just as we would use mlogit, ologit, or oprobit with nonsurvey data.

In our dataset, we have a variable health containing self-reported health status, which takes on the values 1–5, with 1 being "poor" and 5 being "excellent". Since this is an ordered categorical variable, it makes sense to model it using svyolog or svyoprob. As predictors, we use basic demographic variables: female (1 if female, 0 if male), black (1 if black, 0 otherwise), age, and age2 (= age^2):

```
. svyolog health female black age age2

Survey ordered logistic regression

pweight:  finalwgt                          Number of obs   =      10335
Strata:   strata                            Number of strata =         31
PSU:      psu                               Number of PSUs  =         62
                                            Population size = 1.170e+08
                                            F(  4,     28) =     223.27
                                            Prob > F        =     0.0000

------------------------------------------------------------------------------
  health |     Coef.    Std. Err.       t     P>|t|    [95% Conf. Interval]
---------+--------------------------------------------------------------------
  female | -.1615219    .0523678     -3.084   0.004   -.2683266   -.0547171
   black | -.986568     .0790276    -12.484   0.000   -1.147746   -.8253901
     age | -.0119491    .0082974     -1.440   0.160   -.0288717    .0049736
    age2 | -.0003234    .000091      -3.552   0.001    -.000509   -.0001377
---------+--------------------------------------------------------------------
   _cut1 | -4.566229    .1632559    -27.970   0.000   -4.899192   -4.233266
   _cut2 | -3.057415    .1699943    -17.985   0.000   -3.404121   -2.710709
   _cut3 | -1.520596    .1714341     -8.870   0.000   -1.870238   -1.170954
   _cut4 |  -.242785    .1703964     -1.425   0.164   -.5903107    .1047407
------------------------------------------------------------------------------
```

According to our model, females give self-reports of poorer health status than males, blacks report much poorer health status than nonblacks, and older people report worse health than younger.

If we model the categories of self-reported health status as unordered categories using `svymlog`, we get the following results:

```
. svymlog health female black age age2

Survey multinomial logistic regression

pweight:  finalwgt                              Number of obs    =      10335
Strata:   strata                                Number of strata =         31
PSU:      psu                                   Number of PSUs   =         62
                                                Population size  = 1.170e+08
                                                F( 16,     16)   =      36.41
                                                Prob > F         =     0.0000

------------------------------------------------------------------------------
   health |      Coef.   Std. Err.       t    P>|t|     [95% Conf. Interval]
----------+-------------------------------------------------------------------
poor      |
   female | -.1983735    .1072747     -1.849   0.074    -.4171617    .0204147
    black |  .8964694    .1797728      4.987   0.000     .5298203    1.263119
      age |  .0990246    .032111       3.084   0.004     .0335338    .1645155
     age2 | -.0004749    .0003209     -1.480   0.149    -.0011294    .0001796
    _cons | -5.475074    .7468576     -7.331   0.000      -6.9983   -3.951848
----------+-------------------------------------------------------------------
fair      |
   female |  .1782371    .0726556      2.453   0.020      .030055    .3264193
    black |  .4429445    .122667       3.611   0.001     .1927635    .6931256
      age |  .0024576    .0172236      0.143   0.887    -.0326702    .0375853
     age2 |  .0002875    .0001684      1.707   0.098    -.0000559     .000631
    _cons | -1.819561    .4018153     -4.528   0.000    -2.639069   -1.000053
----------+-------------------------------------------------------------------
good      |
   female | -.0458251    .074169      -0.618   0.541    -.1970938    .1054437
    black | -.7532011    .1105444     -6.814   0.000    -.9786579   -.5277443
      age |  -.061369    .009794      -6.266   0.000     -.081344   -.0413939
     age2 |  .0004166    .0001077      3.869   0.001      .000197    .0006363
    _cons |  1.815323    .1996917      9.091   0.000     1.408049    2.222597
----------+-------------------------------------------------------------------
excell    |
   female |  -.222799    .0754205     -2.954   0.006    -.3766202   -.0689778
    black |  -.991647    .1238806     -8.005   0.000    -1.244303   -.7389909
      age | -.0293573    .0137789     -2.131   0.041    -.0574595    -.001255
     age2 | -.0000674    .0001505     -0.448   0.657    -.0003744    .0002396
    _cons |  1.499683    .286143       5.241   0.000     .9160909    2.083276
------------------------------------------------------------------------------
(Outcome health==average is the comparison group)
```

We see an interesting pattern here. It suggests that females are less likely to report the extremes of health than males: females are less likely to report poor health and less likely to report excellent health. The results for blacks, on the other hand, are monotonic: the better the health rating, the less likely they are to report it, relative to nonblacks.

Just like `mlogit`, `svymlog` can display results as relative risk ratios, either at the time of estimation or when redisplaying results:

```
. svymlog, rrr
```
  (output omitted)

At the time of estimation, one can also specify the base category for the comparison:

```
. svymlog health female black age age2, base(1)
```
  (output omitted)

Estimating linear combinations of coefficients with the svylc command

In addition to these new commands, this insert contains an updated version of the `svylc` command, which estimates linear combinations of coefficients. For example, after the first `svymlog` command shown above,

```
. svymlog health female black age age2
```

one might want to estimate the relative risk ratio for nonblack males to black females for the "excellent" category relative to the base "fair" category. This can be done using svylc:

```
. svylc -[excell]female - [excell]black, rrr

( 1) - [excell]female - [excell]black = 0.0

------------------------------------------------------------------------------
  health |      RRR    Std. Err.       t    P>|t|     [95% Conf. Interval]
---------+--------------------------------------------------------------------
     (1) |  3.368427    .4810747     8.503   0.000     2.517245    4.507429
------------------------------------------------------------------------------
```

See the [R] **svylc** entry of the Stata Reference Manual for details on using the svylc command. Note that svyolog and svyoprob, as well as svymlog, require that svylc be used with multiple-equation syntax. This is not at all obvious in the case of svyolog and svyoprob. Indeed, the equation names for svyolog and svyoprob were purposely hidden so that their output looks like that of ologit and oprobit. However, their true equation labeling can be seen by typing svylc, show. The syntax to use with svylc and svytest will then be immediately apparent.

If there are value labels on the dependent variable, svymlog will use these labels for the equation names, just as mlogit does. If any of the value labels contain blanks (e.g., a label "age 70+"), then this will create problems for svylc and svytest, just as it does for the test command. The obvious workaround is not to use value labels with blanks. Note that blanks in value labels cause absolutely no problems for the svymlog command itself.

Methods, formulas, and saved results

The svymlog, svyolog, and svyoprob commands use "linearization"-based variance estimators that are natural extensions of the variance estimator used in svytotal. The point estimates are those given by the "pseudo-maximum-likelihood" estimator; i.e., they are the same point estimates produced by mlogit, ologit, and oprobit when aweights are specified. See [R] **svyreg** and the references therein for details.

The svymlog, svyolog, and svyoprob commands store the same saved results as svylogit and svyprobt (see [R] **svyreg**). In addition, all three commands store in the global macro S_E_ncat the number of categories of the dependent variable. The svymlog command stores in the global macro S_E_base the value of the base category.

Reference

McDowell, A., A. Engel, J. T. Massey, and K. Maurer. 1981. Plan and operation of the Second National Health and Nutrition Examination Survey, 1976–1980. *Vital and Health Statistics* 15(1). National Center for Health Statistics, Hyattsville, MD.

STB categories and insert codes

Inserts in the STB are presently categorized as follows:

General Categories:

an	announcements	*ip*	instruction on programming
cc	communications & letters	*os*	operating system, hardware, &
dm	data management		interprogram communication
dt	datasets	*qs*	questions and suggestions
gr	graphics	*tt*	teaching
in	instruction	*zz*	not elsewhere classified

Statistical Categories:

sbe	biostatistics & epidemiology	*ssa*	survival analysis
sed	exploratory data analysis	*ssi*	simulation & random numbers
sg	general statistics	*sss*	social science & psychometrics
smv	multivariate analysis	*sts*	time-series, econometrics
snp	nonparametric methods	*svy*	survey sampling
sqc	quality control	*sxd*	experimental design
sqv	analysis of qualitative variables	*szz*	not elsewhere classified
srd	robust methods & statistical diagnostics		

In addition, we have granted one other prefix, *stata*, to the manufacturers of Stata for their exclusive use.

Guidelines for authors

The Stata Technical Bulletin (STB) is a journal that is intended to provide a forum for Stata users of all disciplines and levels of sophistication. The STB contains articles written by StataCorp, Stata users, and others.

Articles include new Stata commands (ado-files), programming tutorials, illustrations of data analysis techniques, discussions on teaching statistics, debates on appropriate statistical techniques, reports on other programs, and interesting datasets, announcements, questions, and suggestions.

A submission to the STB consists of

1. An insert (article) describing the purpose of the submission. The STB is produced using plain TeX so submissions using TeX (or LaTeX) are the easiest for the editor to handle, but any word processor is appropriate. If you are not using TeX and your insert contains a significant amount of mathematics, please FAX (409–845–3144) a copy of the insert so we can see the intended appearance of the text.

2. Any ado-files, `.exe` files, or other software that accompanies the submission.

3. A help file for each ado-file included in the submission. See any recent STB diskette for the structure a help file. If you have questions, fill in as much of the information as possible and we will take care of the details.

4. A do-file that replicates the examples in your text. Also include the datasets used in the example. This allows us to verify that the software works as described and allows users to replicate the examples as a way of learning how to use the software.

5. Files containing the graphs to be included in the insert. If you have used STAGE to edit the graphs in your submission, be sure to include the `.gph` files. Do not add titles (e.g., "Figure 1: ...") to your graphs as we will have to strip them off.

The easiest way to submit an insert to the STB is to first create a single "archive file" (either a `.zip` file or a compressed `.tar` file) containing all of the files associated with the submission, and then email it to the editor at `stb@stata.com` either by first using `uuencode` if you are working on a Unix platform or by attaching it to an email message if your mailer allows the sending of attachments. In Unix, for example, to email the current directory and all of its subdirectories:

```
tar -cf - . | compress | uuencode xyzz.tar.Z > whatever
mail stb@stata.com < whatever
```

International Stata Distributors

International Stata users may also order subscriptions to the *Stata Technical Bulletin* from our International Stata Distributors.

Company:	Applied Statistics &
	Systems Consultants
Address:	P.O. Box 1169
	Nazerath-Ellit 17100, Israel
Phone:	+972 66554254
Fax:	+972 66554254
Email:	sasconsl@actcom.co.il
Countries served:	Israel

Company:	Dittrich & Partner Consulting
Address:	Prinzenstrasse 2
	D-42697 Solingen
	Germany
Phone:	+49 212-3390 200
Fax:	+49 212-3390 295
Email:	evhall@dpc.de
Countries served:	Austria, Germany, Italy

Company:	Metrika Consulting
Address:	Mosstorpsvagen 48
	183 30 Taby Stockholm
	Sweden
Phone:	+46-708-163128
Fax:	+46-8-7924747
Email:	hedstrom@metrika.se
Countries served:	Baltic States, Denmark, Finland,
	Iceland, Norway, Sweden

Company:	Ritme Informatique
Address:	34 boulevard Haussmann
	75009 Paris
	France
Phone:	+33 1 42 46 00 42
Fax:	+33 1 42 46 00 33
Email:	info@ritme.com
URL:	http://www.ritme.com
Countries served:	Belgium, France,
	Luxembourg, Switzerland

Company:	Smit Consult
Address:	Doormanstraat 19
	5151 MG Drunen
	Netherlands
Phone:	+31 416-378 125
Fax:	+31 416-378 385
Email:	j.a.c.m.smit@smitcon.nl
URL:	http://www.smitconsult.nl
Countries served:	Netherlands

Company:	Survey Design & Analysis Services P/L
Address:	249 Eramosa Road West
	Moorooduc VIC 3933
	Australia
Phone:	+61 3 5978 8329
Fax:	+61 3 5978 8623
Email:	sales@survey-design.com.au
URL:	http://survey-design.com.au
Countries served:	Australia, New Zealand

Company:	Timberlake Consultants
Address:	47 Hartfield Crescent
	West Wickham
	Kent BR4 9DW U.K.
Phone:	+44 181 462 0495
Fax:	+44 181 462 0493
Email:	info@timberlake.co.uk
URL:	http://www.timberlake.co.uk
Countries served:	United Kingdom, Eire

Company:	Timberlake Consulting S.L.
Address:	C/ Montecarmelo No 36 Bajo
	41011 Seville
	Spain
Phone:	+34.5.428.40.94
Fax:	+34.5.428.40.94
Teléfono móvil:	+34.39.78.64.31
Countries served:	timberlake@zoom.es
	Spain

Company:	Timberlake Consultores
Address:	Praceta do Comércio,
	N° 13–9° Dto. Quinta Grande
	2720 Alfragide Portugal
Phone:	+351 (01) 4719337
Telemóvel:	0931 62 7255
Email:	timberlake.co@mail.telepac.pt
Countries served:	Portugal